Training guide for the c360 SDK for Microsoft CRM4.0

Table of Contents

1. Introduction to the c360 SDK

c360 SDK Training Overview

Overall Course Objectives

By the end of the course, the participant will be able to:

- Install, configure, and license the c360 SDK.
- Build web pages that will integrate with Microsoft CRM and have the same behavior, look and feel.
- Use visual controls for user input that behave, look, and feel like Microsoft CRM (example: DatePicker)
- Fetch data from an external data source (example: bank account) and display it in such a way that it is linked to CRM data.

SDK Training Participant Pre-requisites

It is assumed that the participants in this course:

- Have development experience in C#.
- Have at least some Microsoft CRM **user** experience; Microsoft CRM **developer** experience is preferred.
- Have been encouraged to do prior reading on Microsoft CRM in general before attending c360 SDK training.
- Understand that a single file will be used for the ease of demonstration, even though best practices suggest using separate files for C# code vs. HTML content.

Purpose and Benefits of the c360 SDK

The c360 SDK:

- Simplifies the development of custom, integrated, web-based products to be used in conjunction with Microsoft CRM.
- Ensures the same behavior, look and feel of Microsoft CRM.
- Allows the developer to focus more on business function and logic, and less on form.

Other Notes

The c360 SDK:

- Is web-based, but not designed for Windows applications.
- Is not independent (must have Microsoft CRM).
- Addresses the frustration of developers who do not have a reusable set of visual controls to use in conjunction with Microsoft CRM.
- Will continue to be enhanced and expanded over time.

2. Installing, Licensing and Configuring the c360 SDK

Installation

Clarification

This chapter outlines the details for installing the SDK in a development environment and for the client's on-site deployment as well.

Existing c360 Customers

Some c360 customers may already have components of the c360 SDK installed because it was included with other c360 products, such as Relationship Explorer. **Therefore, there are restrictions on installation locations. Please see below.**

Where can I install the c360 SDK?

The SDK can be installed in any virtual directory of a web server EXCEPT in the same virtual directory of an existing c360 product. Other c360 products may require a specific version of the SDK components to function correctly therefore; these SDK files must remain separate from other c360 products. If the SDK is installed in a sub-directory of a virtual directory, in Web.config, the c360Location key needs to be added and pointed to the location of the installation (see "Post Installation Configuration" on next page).

What version of the .NET framework can I use?

The c360 Solutions SDK can be used with .NET v2.0, .NET 3.0 or .NET 3.5.

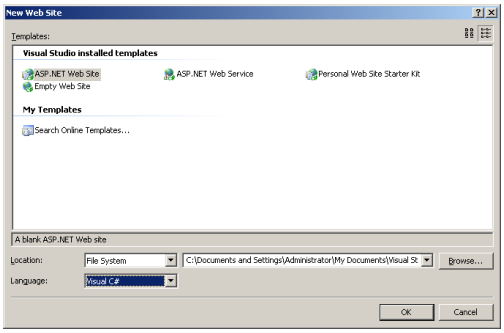
Installation (con't)

Detailed installation Procedure:

1. Download the SDK from our download center: www.c360.com

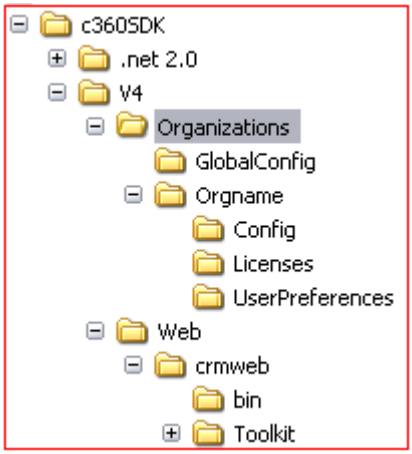
2. Obtain the licenses.
   - If you are using the Adventure Works Cycle demo system, you can download free licenses from our web site: www.c360.com/Licenses.aspx
   - If you are developing on a system with a different organization name, evaluating deploying to a production environment, request evaluation licenses: ww.c360.com/Evaluation.aspx
   - If you are deploying to a production environment, purchase the licenses on our eStore: http://c360.stores.yahoo.net/ and we will email you the licenses automatically. Please allow 24-48 hours.

3. In Visual Studio 2005 create a new "Web site"; select "ASP.NET web site" as your template; select the appropriate language (either C# or VB.NET) and specify the location. Visual Studio 2005 will offer a default name for this site such as "WebSite1", but you want to change this name to something more descriptive, like "c360SdkTest"



4. Extract the SDK zip file to the location where your new web site was created. By default, this will be "C:\Documents and Settings\<user name>\My Documents\Visual Studio 2005\WebSites\c360SdkTest\c360sdk\". You can either extract it to any location on the disk. For example "C:\Program Files\c360sdk\". But make sure that all the files are extracted to the 'c360sdk' folder.

5. The extracted zip file contains the below folder structure.



Installation (con't)

In this structure rename the 'orgname' folder to the Organization name which you intend to work with (this is case sensitive). If you want to work with multiple organizations then create another folder (including sub-folders) with the name of organization name you are working with.

Place the c360.config file in the Config folder and License file in the Licenses folder.

6. Move the exercises out of the .NET 2.0 sub-folder to the root of your new published web site if you are interested in the samples otherwise, it's safe to delete the .NET 2.0 folder.

7. You will have to create a virtual directory named "C360SDK" in Microsoft Dynamics CRM website. If you are working with IIS 7.0 then you will have to create a new applicaction by name 'C360SDK' instead of the virtual directory.

   Creation of C360SDK virtual directory/Application is mandatory because the controls exposed through the c360 SDK refers to the files which are inside the c360SDK virtual directory/Application. This VDIR/Application should point to the above extracted zip files folder. For example if zip file is extracted to "C:\Documents and Settings\<user name>\My Documents\Visual Studio 2005\WebSites\C360SDK\" then VDIR should point to "C:\Documents and Settings\<user name>\My Documents\Visual Studio 2005\WebSites \C360SDk\v4\Web\crmweb\". If you have created an application then add a web.config file to the \crmweb\ folder.

8. Add the "c360Location" key in your Web.config to indicate the name of the virtual directory where you intend to deploy your sdk. See page 16 in the training manual for more details. For example, the key would look like this:

```
<add key="c360Location" value="/c360Sdk/" />
```

9. Add the "c360SDKLocation" key in your c360.config specific to the organization you are working with to indicate the name of the virtual directory where you intend to deploy your SDK. For example, the key would look like this:

```
<add key="c360SDKLocation" value="/c360Sdk/" />
```

10. Earlier versions of CRM provided single organization solutions. CRM 4.0 version lets you to host multiple organizations (Multitenancy) in a single deployment. Each organization must have its own root folder (named for the organization). The root folder should contain a 'config' folder where the c360.config file for the particular organization exists and also a 'licenses' folder which is also specific to the organization.

11. Copy the license you obtained from c360 to the “Licenses” folder inside the specific organizations root folder of your new web site.

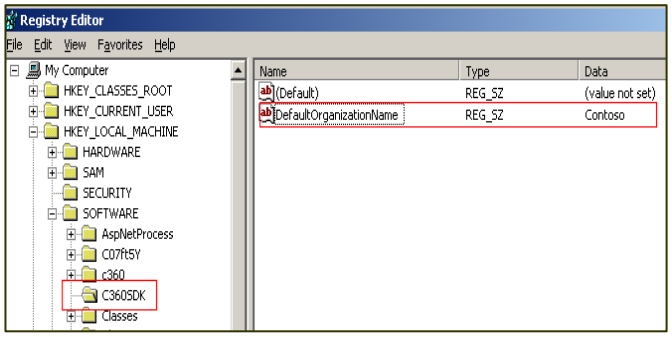
Installation (con’t)

12. Edit the Web.config to disable ASP.NET ‘request validation’ in order to avoid the following error: “A potentially dangerous error Request.Form value was detected from the client”:

```
<system.web>
  <pages validateRequest="false" />
</system.web>
```

13. Create a key ‘C360SDK’ in the registry in the path “HKEY_LOCAL_MACHINE\SOFTWARE”. Add a string ‘DefaultOrganizationName’ under that key this should contain the default organization name of the CRM. This will be used by the SDK to validate the license and fetch the organization specific data. User can override this (in the case of a multi-tenant environment) by providing the organization name in the query string of the URL. For example the url can look like below:

```
http://servername:portnumber/foldername/TestPage.aspx?orgname=<OrganizationName>
```



14. All the controls and pages in the SDK need to be aware of the current organization on which end user is working with. The CRM SDK expects the user to supply the organization. The Organization name supplied will be used to validate the licenses and to fetch organization specific data using CRM web services. In addition to setting the organization name as a querey parameter, the SDK exposes an IO*rganizationProvider* Interface which contains the GetOrganizationName() Method. This method is used by all the pages and controls to identify the Organization Name.
    i. The user has to build an assembly containing a class which Implements IOrganizationProvider Interface from the c360.Toolkit.dll. Whenever the SDK needs the Organization Name, it will call the piece of code that the user has written.

ii. The user has to provide the fully qualified name of the organization provider class and the namespace in Web.config as follows

```
<add key="CustomOrganizationProvider" value="[Namespace.ProviderClassName], [AssemblyName]"/>
```

To find out how to implement the Custom Organization Provider click here

Installation (con't)

15. The client side variable 'ORG_UNIQUE_NAME' contains the current organization name the user is working with.This can be used in the client side scripts to attach the orgname to the query parameter. This can be used in Ifrmaes and other controls.

16. The server-side variable 'c360.Toolkit.Utility.c360UserContext.CurrentOrganizationName' contains the organization name the user is working with. This variable is set on the first call to any c360SDK page, and can be used on subsequent calls to attach the orgname to the query parameter in the server side code.

17. You have to create a user defined role called "c360 Metadata Reader" – (recommended name for metadata reader role) and add all users to join that role in order for metadataservice to fetch metadata for non admin users. If you already have any of the c360 products installed in the machine then no need to create the role since the role already exists. **You just need to add the user to the role.**

    The following privileges should be set for the role.

    - Read privilege to Entity, Attribute, Relationship, E-mail Template, Article Template and Contract Template.
    - Create privilege to ISV Extensions, Import Customizations, Export Customization and Publish Customizations.

Licensing

Please contact your c360 account manager (contact sales@c360.com if you do not know who your account manager is) to request a license file that will allow you to use the c360 SDK in your custom application. When requesting licenses, you must provide the number of active CRM licenses and the Organization Name of the system where the c360 SDK will be used. Upon receiving this information, c360 will send you a file called **c360.toolkit.sdk Version 4.lic**. You must save this file in the **Licenses** folder created during installation.

Post-Installation Configuration (Optional)

As of this writing, fourteen (14) optional settings can be configured in the c360.config file for specific organization inside the "appSettings" section.

This document lists all the optional settings that can be added to the c360.config file of the specific organization to modify the SDK default configuration.

Here is an example:

```
<add key='CaseSensitiveNameOfTheKey' value='TheValue'/>
```

Setting	Description/Comment
CrmUrl	Indicates the URL of the CRM website. This key is necessary whenever you are developing your SDK solution outside of the CRM website. So, for example if you are developing locally or using dynamic ports when previewing your pages with Visual Studio 2005. Without this key, attempting to open a CRM record will result in a 404 error. `<add key="CrmUrl" value="http://localhost:5555" />`
Language	The site wide language for all c360 products. Sample: `<add key="Language" value="English"/>`
ReportServicesUrl	Indicates where the Microsoft SQL Reporting Server services are available. This is useful when SRS is on a different machine or on a different TCP port or on a HTTPS site for example. Sample: `<add key="ReportServicesUrl" value="https://servername:7000/ReportServer"/>`

CrmDataBaseConnectionString	The connection string to the Microsoft CRM database.
	
	Windows integrated security sample:
	`<add key="CrmDataBaseConnectionString" value="SERVER=myServer;DATABASE=Microsoft_CRM_MSCRM;Trusted_Connection=yes"/>`
	
	Native SQL security sample:
	`<add key="CrmConnectionString" value="Data Source=sqlServerName;Initial Catalog=Microsoft_CRM_MSCRM;User ID=sa;Password=saPassword"/>`

When CRM and SQL are installed on the same machine, we recommend using a connection string with integrated Windows security. However, when they are installed on different machines, we strongly recommend using a connection string with native SQL security to avoid 'credentials delegation' problems. There are numerous articles on c360's support web site and on Microsoft's support web site that describe this delegation problem and offer various suggestions to solve it, but the best way to avoid the problem in the first place is to use SQL native security.

Post-Installation Configuration (con't)

Setting	Description/Comment
CacheTimeoutInMinutes	Indicates how long values that are not dependent on a file, such as the current user's credentials, will be cached in the web application's memory. The default value is 10 minutes. Sample: <add key="CacheTimeoutInMinutes" value="5"/>
DataBaseConnectionString	The connection string to the c360 database. Windows integrated security sample: <add key="DataBaseConnectionString" value="SERVER=myServer;DATABASE=c360_Solutions;Trusted_Connection=yes"/> Native SQL security sample: <add key="DataBaseConnectionString" value="Data Source=sqlServerName;Initial Catalog=c360_Solutions;User ID=sa;Password=saPassword"/>
LogHttpTraffic	This key is used to indicate whether HTTP Traffic should be logged into the Event viewer <add key="LogHttpTraffic" value="false"/>
LogHttpPostDetails	This value indicates whether Form info should be logged into the Event viewer <add key= "LogHttpPostDetails" value="false"/>
LogHttpHeaderInfo	This value indicates whether Header info including cookie details should be logged into the Event viewer <add key= "LogHttpHeaderInfo" value="false"/>
DiscoveryWebServicesUrl	This key is present in the c360GlobalConfig.config file in the GloablConfig folder. Update this key with the DiscoveryServiceUrl if your application and platform server are installed in different machines. Sample: <add key="DiscoveryWebServicesUrl" value="http://servername:8000/MSCRMServices/"/>
Administration.Identity.Username	This key is used to impersonate the web service calls. But this is not applicable for SDK.
Administration.Identity.Domain	This key is used to impersonate the web service calls. But this is not applicable for SDK.
Administration.Identity.Password.Encrypted	This key is used to impersonate the web service calls. But this is not applicable for SDK.

Post-Installation Configuration (Mandatory)

Setting	Description/Comment
c360SDKLocation	This key should be added to the c360.config file. This key indicates the name of the virtual directory where the c360 sdk is installed. `<add key="c360SDKLocation" value="/c360sdk/"/>`

c360Location	The one and only key that can be added to the web.config file. This key is required. This must point to the installation location of the c360 SDK relative to the root of the website. If it is installed in a independent virtual directory , then the value would be:
	`<add key="c360Location" value="/c360sdk/"/>`
	
	If it was installed into a sub-directory 'c360sdk' of the website, for example, 'c360sdktest', the value would be:
	`<add key="c360Location" value="/c360sdktest/c360sdk/v4/web/crmweb/"/>`

Organization Provider

How do I provide an Organization Provider?

You have to implement the IOrganization Provider Interface from the c360.Toolkit.dll and implement the method (GetOrganizationName) which will returns the Organization Name. This provides another way to support multi-tenancy besides appending the organization name to the URL as a query parameter.

```
using System.Web;
using c360.Toolkit.OrganizationProvider;

namespace C360.CustomOrgProvider
{
        public class TextOrgProvider: IOrganizationProvider
        {
                public string GetOrganizationName (HttpContext PageContext)
                {
                                return "myOrg";
                }
        }
}
```

For the above Implementation, the entry in web.config goes like this:

```
<add key="CustomOrganizationProvider" value="C360.CustomOrgProvider.TextOrgProvider, C360.CustomOrgProvider"/>
```

Online Resources

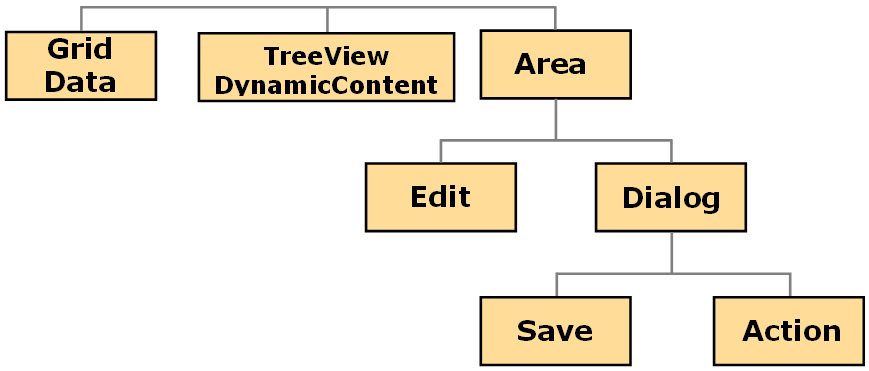
Go to www.c360.com/Forums to become part of the online community of developers using the c360 SDK for Microsoft CRM v4.0.

3. Creating Web Pages with the c360 SDK

Overview of c360 SDK Structure

To create pages utilizing the c360 SDK, you will derive your page from one of the six base classes listed below, and then add your desired c360 SDK Controls to that page, and some logic to handle the data.

The c360 SDK has six (6) base classes, or templates, that define the look and feel of the web page you create. There are two (2) types of dialogs.



Web Page	Description/Comment
GridDataPage	• Used to fetch data to be displayed in a grid • Derives from Base • No visual HTML
TreeView DynamicContent	• Used to fetch data to be displayed in a tree view • No visual HTML • Derives from Base
AreaPage	• What you build onto • Derives from Base • Most used Web Page for Microsoft CRM custom development (>75%)
EditPage	• Provides easy way to save user input • Derives from AreaPage • Equivalent of Microsoft CRM Edit web page • Includes a menu bar and a button bar • Left area and footer are hidden (developer can override to display them)
DialogPage	• Useful for pop-ups, user input; two types: • SavePage dialog: prompts user to save • ActionPage dialog: Performs save actions on user-selected records in a grid • Derives from AreaPage

Base Class

Overview

The Base class is inherited by every page in the c360 SDK, and has properties and methods that are available to all of the pages in the c360 SDK.

Base Properties

Property (type)	**Description/Comment**
CrmService[1] (ICrmService)	Reference to an object allowing you to call most methods of the CRM API.
CurrentUser (ISecurityPrincipal)	Reference to an object containing information about the current user.
MetadataService (IMetadataService)	Reference to an object allowing you to call most methods of the CRM metadata API.
ObjId (String)	The GUID of the entity (if applicable) passed by the Microsoft CRM application.
ObjMetadata (EntityMetadata)	Metadata regarding the entity passed by the Microsoft CRM application. Please note that if no entity type is passed from the CRM application, this property will be *null*.
ObjType (String)	The type of the entity (if applicable) passed by the Microsoft CRM application.
PageTitle (String)	Allows developer to get or set the current page title.

Base Methods

Method (return)	**Description/Comment**
Impersonate (Void)	Impersonates a user different than the currently logged in user. To successfully impersonate a different user, this method must be called prior to the OnInit event is fired since this is when the user is normally authenticated.

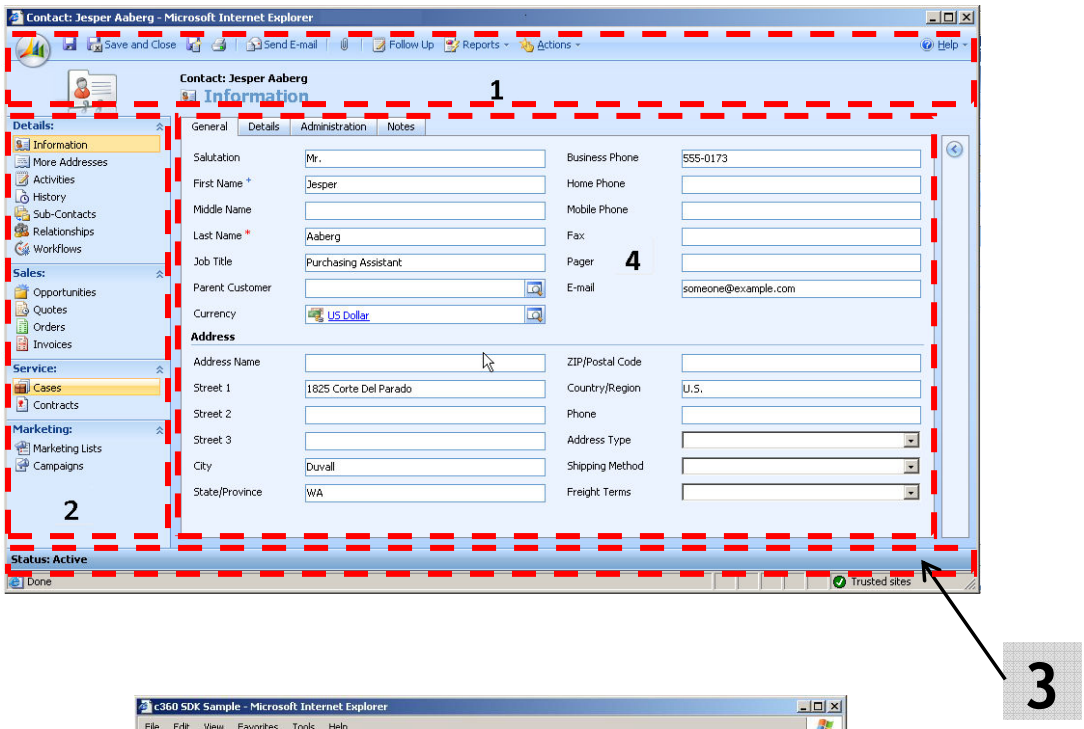
[1] Using this object avoids the need to create a reference to the Microsoft CRM web services and to instantiate an object every time you need to invoke a CRM API method. This property is available for your convenience but keep in mind that this has some limitations. The most important one is being the web service "proxy" which was generated in c360's development lab by the time the SDK was released and therefore it is not aware of custom fields.
For strongly typed objects that are aware of custom fields that you added to your CRM system, you will need to include a reference to your web services and generate your own "proxy".

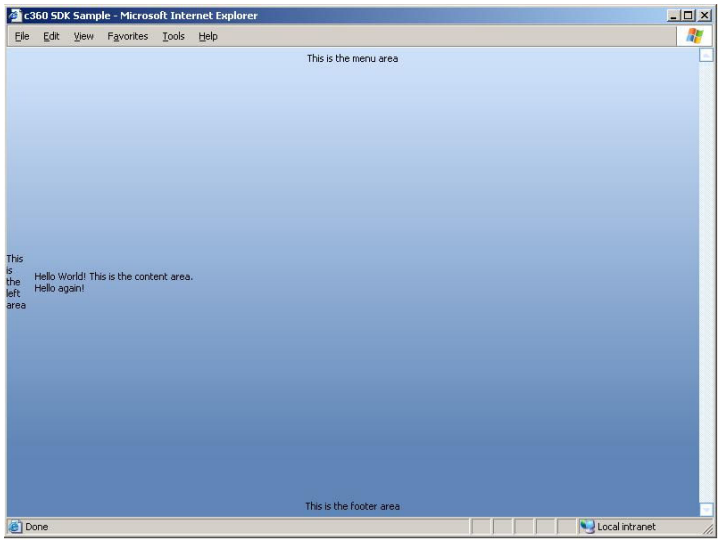
AreaPage

Overview

This page class enforces the CRM look and feel, and is the most used. It takes care of including the necessary style sheets and most importantly, it creates the 4 areas that are always present in a CRM web page:

1. **Menu area** (typically light blue background)
2. **Left navigation area** (typically light blue)
3. **Footer area** (typically displays Status, etc.)
4. Actual content





You always have the ability to alter the width or height of any these areas. Setting the width or height to zero (0) will remove that area from the resulting HTML.

AreaPage (cont'd)

AreaPage Properties

Property (type)	Description/Comment
ContentArea (TableCell)	The main content area. You can programmatically add content to this object like this: `Instance.ContentArea.Controls.Add(new LiteralControl('This text displayed in Content Area'));` Any HTML you have in the ASPX page will be automatically added inside of this region.
FooterArea (TableCell)	The bottom area where the area links are displayed. You can programmatically add content to this object like this: `Instance.FooterArea.Controls.Add(new LiteralControl('This text displayed in Footer Area'));`
Form (Htlmform)	Reference to the HTML form included in the page.
Instance (IAreaPage)	Reference to the current page. You should always refer to this property to ensure future compatibility. For example, we recommend that you use the following syntax… `Instance.PageTitle = "testing 123";` …instead of the following syntax: `Instance.PageTitle = "testing 123";`
LeftArea (TableCell)	The left area where the navigation menu is displayed. You can programmatically add content to this object like this: `Instance.LeftArea.Controls.Add(new LiteralControl('This text displayed in Left Area'));`
MenuArea (TableCell)	The top area where the menu is displayed. You can programmatically add content to this object like this: `Instance.MenuArea.Controls.Add(new LiteralControl('This text displayed in Menu Area'));`
MetaTags (String)	Metatags that will be included in the resulting HTML page. *(Read Only)*
StyleBlocks (String)	Styles that will be included in the resulting HTML page. *(Read Only)*
StyleSheets (String)	Links to style sheets that will be included in the resulting HTML page. *(Read Only)*

AreaPage (cont'd)

AreaPage Methods

Method (return)	**Description/Comments**
IsClientScriptRegistered (bool)	Checks if a certain script is already registered in the current page.
IsMetaTagRegistered (bool)	Checks if a certain meta tag is already registered in the current page.
IsStyleBlockRegistered (bool)	Checks if a certain style block is already registered in the current page.
IsStyleSheetRegistered (bool)	Checks if a certain style sheet is already registered in the current page.
RegisterClientScriptBlock (void)	Adds style information to the resulting HTML.
RegisterClientScriptEvent (void)	Registers a JavaScript function to be called automatically when an event occurs.
RegisterClientScriptFile (void)	Registers an external JavaScript file to be included in the resulting HTML.
RegisterMetaTag (void)	Adds meta tags to the resulting HTML.
RegisterStyleBlock (void)	Adds style information to the resulting HTML.
RegisterStyleSheet (void)	Registers an external style sheet to be included in the resulting HTML.

AreaPage (cont'd)

AreaPage Events

Event (return)	Description/Comments
BuildContentArea (void)	This event is raised when the page is ready to let you add content to the ContentArea. The event allows you to programmatically add HTML controls that will be rendered inside the content area.
BuildFooterArea (void)	This event is raised when the page is ready to let you add content to the FooterArea. The event allows you to programmatically add HTML controls that will be rendered inside the footer area. Please note: this event is not raised if you have previously set the height of the footer area to 0 because a height of zero indicates that you do not want the footer area to be displayed. Therefore, the page does not need to add content to this area.
BuildLeftArea (void)	This event is raised when the page is ready to let you add content to the LeftArea. The event allows you to programmatically add HTML controls that will be rendered inside the left area. Please note: this event is not raised if you have previously set the width of the left area to 0 because a width of zero indicates that you do not want the left area to be displayed. Therefore, the page does not need to add content to this area.
BuildMenuArea (void)	This event is raised when the page is ready to let you add content to the MenuArea. The event allows you to programmatically add HTML controls that will be rendered inside the menu area. Please note: this event is not raised if you have previously set the height of the menu area to 0 because a height of zero indicates that you do not want the menu area to be displayed. Therefore, the page does not need to add content to this area.

AreaPage (cont'd)

How do I add content to the four areas?

As stated in the previous section, you can programmatically add content to any of the four areas by adding a web control to the controls collection of the corresponding area like in the following example:

```
Instance.LeftArea.Controls.Add(new LiteralControl("testing... 123..."));
```

You can add any control that derives from the .NET control class which includes all the standard .NET controls (such as HtmlTable, LiteralControl, etc.), all of the controls provided to you by the SDK, and also the vast majority of the third party controls. In addition to this, be aware that any HTML present in the ASPX page will be automatically added to the content area. For example, if you have <table><tr><td>testing</td></tr></table> in your aspx, this HTML table will be rendered inside the Content area.

How do I change the size of an area?

First of all, it's important to note that you need to change the size of the areas before they are ready to be rendered because when a control is rendered, it's too late to change its properties. What this means is that you can't change the size of the areas from the four events that were presented on the previous page. The proper way to change the size of an area is by the width or the height property of the corresponding area in the constructor of your page class.

The following example shows how you can make the LeftArea very wide and the FooterArea very tall:

```
public MyCustomPage() : AreaPage() {
   Instance.LeftArea.Width = Unit.Pixel(300);
   Instance.FooterArea.Height = Unit.Pixel(250);
}
```

When you change the height of the MenuArea or the FooterArea or when you change the width of the LeftArea, the ContentArea height and width will automatically adjust to fill the entire page.

How do I hide an area?

You can completely hide a section by setting its width or height to zero (0) pixels. Please note: sections are in fact not rendered at all when you set their width or height to zero which means that their content is also not rendered and the corresponding "Build" event mentioned in the previous section is not fired. The following example shows how you can hide the MenuArea:

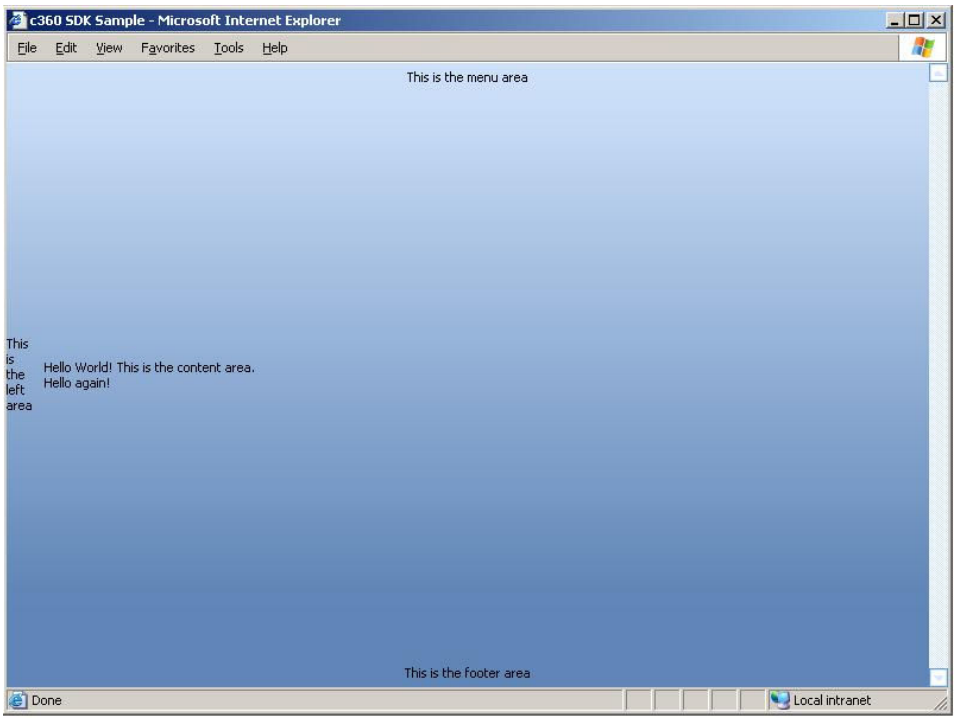
```
public MyCustomPage() : AreaPage() {
   Instance.MenuArea.Height = Unit.Pixel(0);
}
```

AreaPage (cont'd)

Sample Code

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" %>

<script runat="server">
	public override void BuildMenuArea() {
		Instance.MenuArea.Controls.Add(new System.Web.UI.LiteralControl("<center>This is the menu area</center>"));
		Instance.ContentArea.Controls.Add(new System.Web.UI.LiteralControl("<br>Hello again!"));
		Instance.FooterArea.Controls.Add(new System.Web.UI.LiteralControl("<center>This is the footer area</center>"));
		Instance.LeftArea.Controls.Add(new System.Web.UI.LiteralControl("This is the left area"));
	}
</script>
<html>
<head>
	<title>c360 SDK Sample</title>
</head>
<body>
	Hello World! This is the content area.
</body>
</html>
```

Sample Output



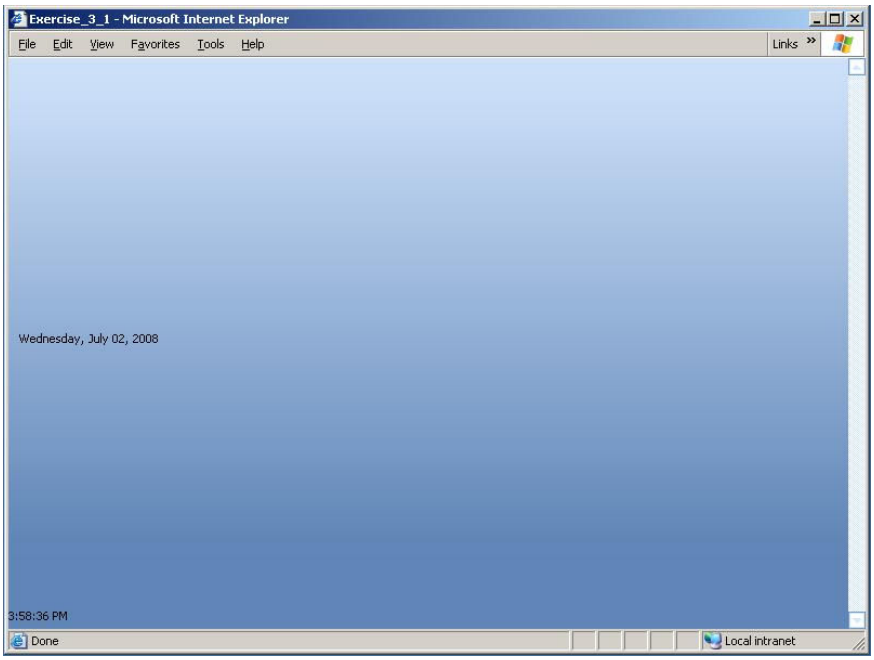
AreaPage (cont'd)

Exercise 3.1

Build a page deriving from AreaPage which:

- does not have a menu area
- does not have a left area
- displays the current date in the content area
- displays the current time in the footer

Exercise 3.1 Desired Output



Exercise 3.1 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" %>

<script runat="server">
        public override void BuildMenuArea() {
                Instance.MenuArea.Height = Unit.Pixel(0);
                Instance.LeftArea.Width = Unit.Pixel(0);
                Instance.ContentArea.Controls.Add(new System.Web.UI.LiteralControl("<br>" + DateTime.Now.ToLongDateString()));
                Instance.FooterArea.Controls.Add(new System.Web.UI.LiteralControl(DateTime.Now.ToLongTimeString()));
        }
</script>
<html>
<head>
          <title>Exercise 1</title>
</head>
<body>
</body>
</html>
```

AreaPage (cont'd)

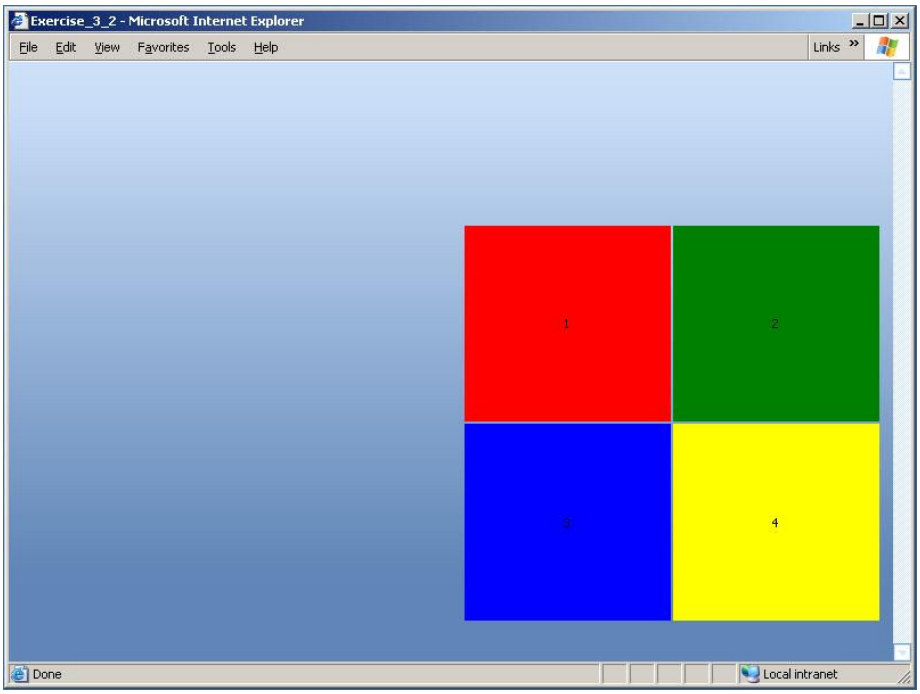
Exercise 3.2

Build a page deriving from AreaPage which:

- has a tall menu area (one quarter of the page height)
- has a wide left area (one half of the page width)
- has a tall footer area (one quarter of the page height)
- displays a two row, two columns table where the four cells each have a different background color

NOTE: Use HTML to build the table

Exercise 3.2 Desired Output



AreaPage (cont'd)

Exercise 3.2 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" %>

<script runat="server">
	public override void BuildMenuArea() {
		Instance.MenuArea.Height = Unit.Percentage(25);
		Instance.LeftArea.Width = Unit.Percentage(50);
		Instance.FooterArea.Width = Unit.Percentage(50);
	}
</script>
<html>
<head>
	<title>Exercise 2</title>
</head>

<body>
	<table width="100%" height="100%">
		<tr>
		<td style="background-color:red;" align="middle">1</td>
		<td style="background-color:green;" align="middle">2</td>
		</tr>
		<tr>
		<td style="background-color:blue;" align="middle">3</td>
		<td style="background-color:yellow;" align="middle">4</td>
		</tr>
	</table>
</body>

</html>
```

AreaPage (cont'd)

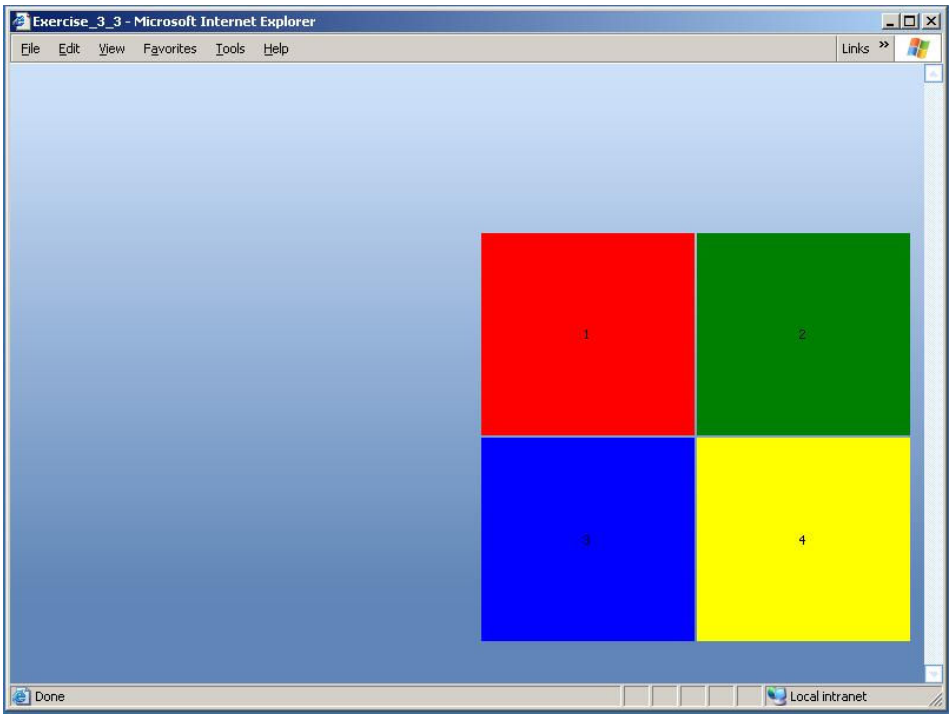
Exercise 3.3

Build a page deriving from AreaPage which:

- has a tall menu area (one quarter of the page height)
- has a wide left area (one half of the page width)
- has a tall footer area (one quarter of the page height)
- displays two row, two columns table where the four cells each have a different background color

NOTE: Use Server side objects to build the table

Exercise 3.3 Desired Output



AreaPage (cont'd)

Exercise 3.3 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" %>

<script runat="server">
        public override void BuildContentArea() {
                Instance.MenuArea.Height = Unit.Percentage(25);
                Instance.LeftArea.Width = Unit.Percentage(50);
                Instance.FooterArea.Width = Unit.Percentage(50);

                HtmlTable objTable = new HtmlTable();
                objTable.Width = "100%";
                objTable.Height = "100%";

                objTable.Rows.Add(new HtmlTableRow());
                objTable.Rows[0].Cells.Add(new HtmlTableCell());
                objTable.Rows[0].Cells.Add(new HtmlTableCell());

                objTable.Rows.Add(new HtmlTableRow());
                objTable.Rows[1].Cells.Add(new HtmlTableCell());
                objTable.Rows[1].Cells.Add(new HtmlTableCell());

                objTable.Rows[0].Cells[0].BgColor = "red";
                objTable.Rows[0].Cells[1].BgColor = "green";
                objTable.Rows[1].Cells[0].BgColor = "blue";
                objTable.Rows[1].Cells[1].BgColor = "yellow";

                objTable.Rows[0].Cells[0].InnerText = "1";
                objTable.Rows[0].Cells[1].InnerText = "2";
                objTable.Rows[1].Cells[0].InnerText = "3";
                objTable.Rows[1].Cells[1].InnerText = "4";

                objTable.Rows[0].Cells[0].Align = "Middle";
                objTable.Rows[0].Cells[1].Align = "Middle";
                objTable.Rows[1].Cells[0].Align = "Middle";
                objTable.Rows[1].Cells[1].Align = "Middle";

                Instance.ContentArea.Controls.Add(objTable);
        }
</script>
<html>
<head>
        <title>Exercise 3</title>
</head>
 <body>
 </body>
 </html>
```

ActionPage

Overview

The ActionPage class is used in conjunction with the grid, and allows the developer to easily create a custom action in a grid menu bar to apply an action on records selected by the user. This class allows the developer to prompt the user to confirm the execution of this action against the selected record and optionally prompt the user for additional data. When the user clicks the "OK" button to confirm this action, the screen is replaced with a progress bar that gets updated as the action is executed against each record.

ActionPage Properties

Property (type)	**Description/Comment**
DialogDescription (string)	(See the "DialogPage" for more details).
DialogTitle (string)	(See the "DialogPage" for more details).
ClientSidePreSaveHandler (string)	The name of a JavaScript method that will be called after the data has been successfully validated. The string returned by this method will be sent to the server-side "SaveData" event. `Instance.ClientSidePreSaveHandler = "jsSaveHandler";`
ClientSideSaveErrorHandler (string)	The name of a JavaScript method that will be called if the save operation fails. Allows you to display an error message and/or perform additional logic. `Instance.ClientSideSaveErrorHandler = "jsErrorHandler";`
ClientSideSaveSuccessHandler (string)	The name of a JavaScript method that will be called if the save operation succeeds. Allows you to display an error message and/or perform additional logic. `Instance.ClientSidePreSaveHandler = "jsSuccessHandler";`
ClientSideValidationHandler (string)	The name of a JavaScript method that will be called when the data on the screen needs to be validated. The JavaScript method should return true to continue processing, or false to prevent the save action from completing. `Instance.ClientSideValidationHandler = "jsValidationHandler";`
DialogHeight (int)	Read-only property that indicates the height of the action window. `int windowHeight = Instance.DialogHeight;`

ActionPage (cont'd)

ActionPage Properties (cont'd)

Property (type)	Description/Comment
DialogWidth (int)	Read-only property that indicates the width of the action window. `int windowWidth = Instance.DialogWidth;`
RecordCount (int)	Read-only property that indicates the total number of records selected by the user and must eventually be processed by this dialog window. `int totalRecordcount = Instance.RecordCount;`
SelectedRecords (XmlNodeList)	Read-only property that contains a list of xml nodes representing each record that the user has selected. `XmlNodeList selectedRecords = Instance.SelectedRecords;`

ActionPage Methods

Method (return)	Description/Comment
AddCancelButton (void)	Adds a "Cancel" button to the bottom section of the window. The caption is automatically added and also translated in the appropriate language. The window is automatically closed when the user clicks on this button. `Instance.AddCancelButton();`
AddOkButton (void)	Adds a "Ok" button to the bottom section of the window. The caption is automatically added and also translated in the appropriate language. When the user clicks on this button, the server-side "SaveData" method is invoked once for every selected record. `Instance.AddOkButton();`

ActionPage (cont'd)

ActionPage Event

Method (return)	**Description/Comment**
SaveData (string)	This event is raised on the server-side once for every selected record after the user clicks on the "Ok" button. This event receives three parameters: 1. The unique identifier of the current record 2. The object type of the selected record 3. A string parameter which contains the value returned by the client-side saves handler function. This method can return a string containing any error messages or an empty string when it completes successfully.

Exercise

We will provide examples of the ActionPage class later in this training, because it relies on the Grid control, we will go over this functionality after looking at this control.

DialogPage

Overview

The DialogPage class is used by pages to be presented in a dialog window. This window is divided into three sections:

- The top third of the page is reserved for a title and description
- The middle section is used to ask the user to confirm a certain action before it gets executed
- The bottom section holds the "OK" and "Cancel" buttons

DialogPage Properties

Property (type)	**Description/Comment**
DialogDescription (string)	The description that appears below the title in smaller text intended to offer the user an explanation of why he is presented with the window. `Instance.DialogDescription = "Click the OK button is you want to deactivate this user.";`
DialogTitle (string)	The information to be presented on the top third of the page in an alternate color and larger font. `Instance.DialogTitle = "Deactivate User";`

DialogPage Methods

Method (return)	**Description/Comment**
AddButton (void)	Adds a button to the bottom section of the window with the caption specified. You can specify the title of the button and also the JavaScript to be executed when the user clicks on the button. `Instance.AddButton("Button 1", "btnMyButton", "jsCustomButtonClick()");`
AddCancelButton (void)	Adds a "Cancel" button to the bottom section of the window. You can specify the JavaScript to be executed when the user clicks on the button but you cannot specify the caption. The caption is automatically added and also translated in the appropriate language. `Instance.AddCancelButton("btnCancelButton", "jsCancelButtonClick()");`

DialogPage (cont'd)

DialogPage Methods (cont'd)

Method (return)	**Description/Comment**
AddDeleteButton (void)	Adds a "Delete" button to the bottom section of the window. You can specify the JavaScript to be executed when the user clicks on the button but you cannot specify the caption. The caption is automatically added and also translated in the appropriate language. `Instance.AddDeleteButton("btnDeleteButton", "jsDeleteButtonClick()");`
AddOkButton (void)	Adds a "Ok" button to the bottom section of the window. You can specify the JavaScript to be executed when the user clicks on the button but the caption is automatically added and also translated in the appropriate language. `Instance.AddOkButton("btnOkButton", "jsConfirmButtonClick()");`

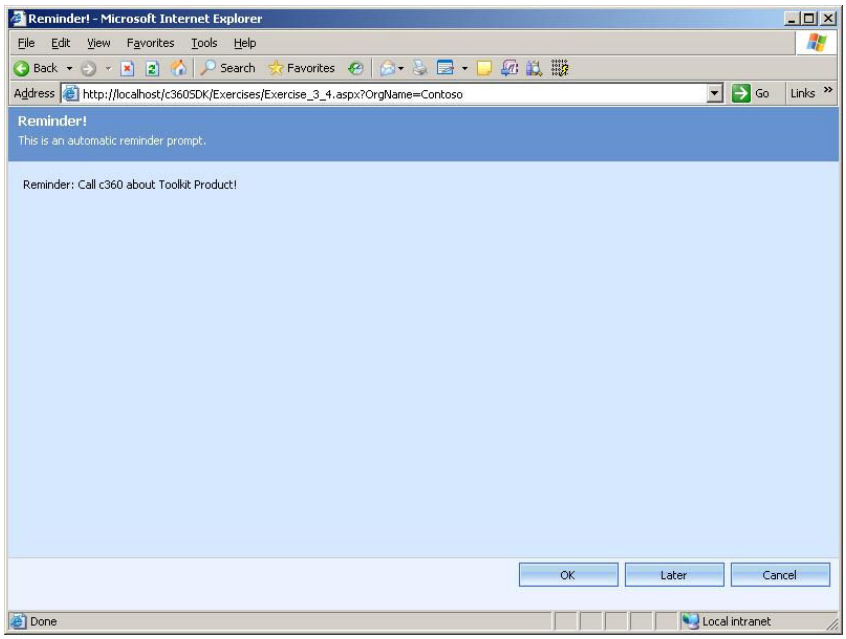
DialogPage (cont'd)

Exercise 3.4

Create a page deriving from DialogPage which:

- Has a dialog title of "Reminder"
- Has a dialog description of "This is an automatic reminder prompt."
- Contains the text "Reminder: Call c360 about Toolkit Product!" in the content area.
- Contains three buttons: OK, Later, and Cancel
  - OK and Cancel buttons call a client script to close the window
  - Later button alerts the user that they will be reminded in 1 day

Exercise 3.4 Desired Output



DialogPage (cont'd)

Exercise 3.4 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.DialogPage" %>
<script runat="server">
        public override void BuildMenuArea() {
                Instance.DialogTitle = "Reminder!";
                Instance.DialogDescription = "This is an automatic reminder prompt.";
                base.BuildMenuArea();
        }

        public override void BuildContentArea() {
                Instance.ContentArea.Controls.Add(new System.Web.UI.LiteralControl("Reminder: Call c360 about Toolkit Product!"));
        }

        public override void BuildFooterArea() {
                Instance.AddOkButton("btnOk", "closeIt()");
                Instance.AddButton("Later","btnLater", "setReminder()");
                Instance.AddCancelButton("btnCancel", "closeIt()");
        }
</script>

<script language="javascript">
function closeIt() {
        window.close();
}

function setReminder() {
        alert("You will be reminded in 1 day(s)");
}

</script>
```

EditPage

Overview

This class is used to build a page where the user can change and ultimately save some data. This class automatically adds a menu bar at the top of the page with a "File" and a "Help" menu and a button bar with the "Save", "Save and New" and the "Save and Close" buttons.

EditPage Properties

Property (type)	Description/Comment
ButtonMenu (IMenu)	Reference to the menu bar with the "Save", "Save and New", and "Save and Close" buttons. You can add custom buttons like so: `Instance.ButtonMenu.Items.Add(ControlFactory.CreateMenuItem("User Preferences"));`
ClientSidePreSaveHandler (string)	The name of a JavaScript method that will be called after the data has been successfully validated. The string returned by this method will be sent to the server-side "SaveData" event. `Instance.ClientSidePreSaveHandler = "jsSaveHandler";`
ClientSideSaveErrorHandler (string)	The name of a JavaScript method that will be called if the save operation fails. Allows you to display an error message and/or perform additional logic. `Instance.ClientSideSaveErrorHandler = "jsErrorHandler";`
ClientSideSaveSuccessHandler (string)	The name of a JavaScript method that will be called if the save operation succeeds. Allows you to display an error message and/or perform additional logic. `Instance.ClientSidePreSaveHandler = "jsSuccessHandler";`
ClientSideValidationHandler (string)	The name of a JavaScript method that will be called when the data on the screen needs to be validated. The JavaScript method should return true to continue processing, or false to prevent the save action from completing. `Instance.ClientSideValidationHandler = "jsValidationHandler";`
DisplayFileMenu (bool)	Specifies if the file menu is displayed or not. `Instance.DisplayFileMenu = false;`
DisplayButtonMenu (bool)	Specifies if the button menu is displayed or not. `Instance.DisplayButtonMenu = false;`

EditPage (cont'd)

EditPage Properties (cont'd)

Property (type)	Description/Comment
DisplayTitleArea (bool)	Specifies if the title area is displayed or not. `Instance.DisplayTitleArea = false;`

FileMenu (IMenu)	Reference to the Menu bar object with the "file" and "help" items. You can add custom pull down menus like so:
	`Instance.FileMenu.Items.Add(ControlFactory.CreateMenuItem("View"));`
	You can also add menu items to existing pull-down menus like so:
	`Instance.FileMenu.Items[0].Items.Add(ControlFactory.CreateMenuItem("Save As"));`

IncludeStandardFileMenuItems (bool)	Specifies if the file menu is to include the standard "File" and "Help" pull-down menus. `Instance.IncludeStandardFileMenuItems = false;`
IncludeStandardButtons (bool)	Specifies if the button bar includes the standard "Save", "Save and Close" and "Save and New" buttons. `Instance.IncludeStandardFileMenuItems = false;`
Status (string)	The string to display in the footer area.

EditPage Event

Method (return)	Description/Comment
SaveData (string)	This event is raised on the server-side when the user has clicked on the "Save", "Save and Close" or "Save and New" buttons. This event receives one string parameter which contains the value returned by the client-side save handler function. This method can return a string containing any error messages or an empty string when it completes successfully.

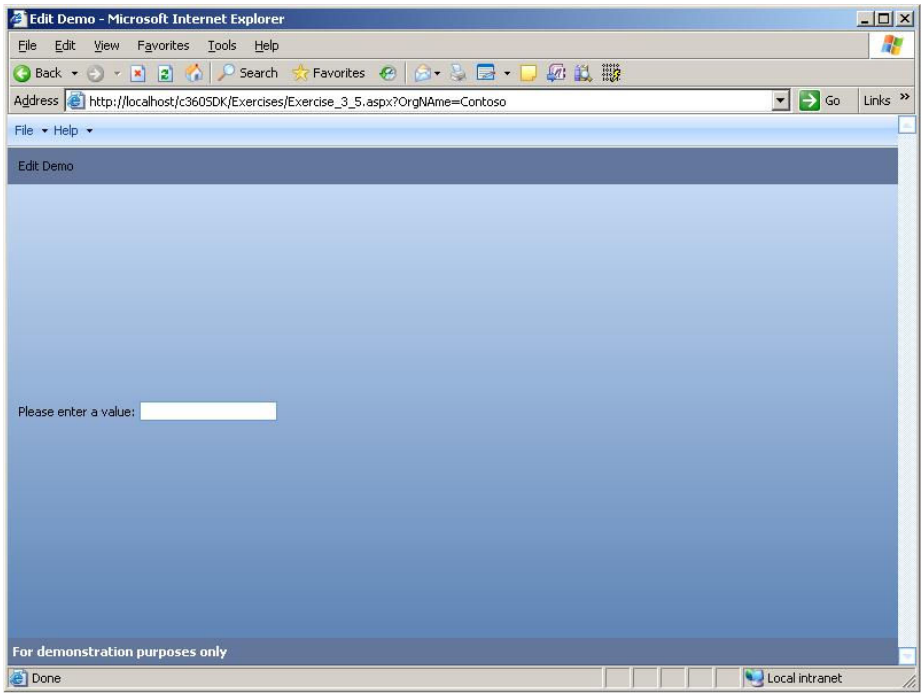
EditPage (cont'd)

Exercise 3.5

Build a page deriving from EditPage which:

- Has a filemenu
- Has no button menu
- Has an ASP.NET web form control text field with the label "Please enter a value"
- Has a page title of "Edit Demo"
- Has a status of "For demonstration purposes only"
- Validates the user input and prevents the save operation to continue if the textbox is empty
- Handles the "SaveData" event

Exercise 3.5 Desired Output



EditPage (cont'd)

Exercise 3.5 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.EditPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildMenuArea() {
                // Define the Title and Status of the EditPage
                Instance.PageTitle = "Edit Demo";
                Instance.Status = "For demonstration purposes only";

                // Hide the button bar
                Instance.DisplayButtonMenu = false;

                // Define the client-side handlers
                Instance.ClientSideValidationHandler = "validate";
                Instance.ClientSidePreSaveHandler = "submit";

                // Continue to build the menu area
                base.BuildMenuArea();
        }

        public override void BuildContentArea() {
                // Add the label and textbox
                Instance.ContentArea.Controls.Add(new System.Web.UI.LiteralControl("Please enter a value: "));
    System.Web.UI.WebControls.TextBox textBox1 = new TextBox();
                textBox1.ID = "tb1";
                Instance.ContentArea.Controls.Add(textBox1);
        }

  public override string SaveData(string stringToSave) {
    // TODO: save the data
    //...

    // Return success indicator
    return "";
  }
</script>
```

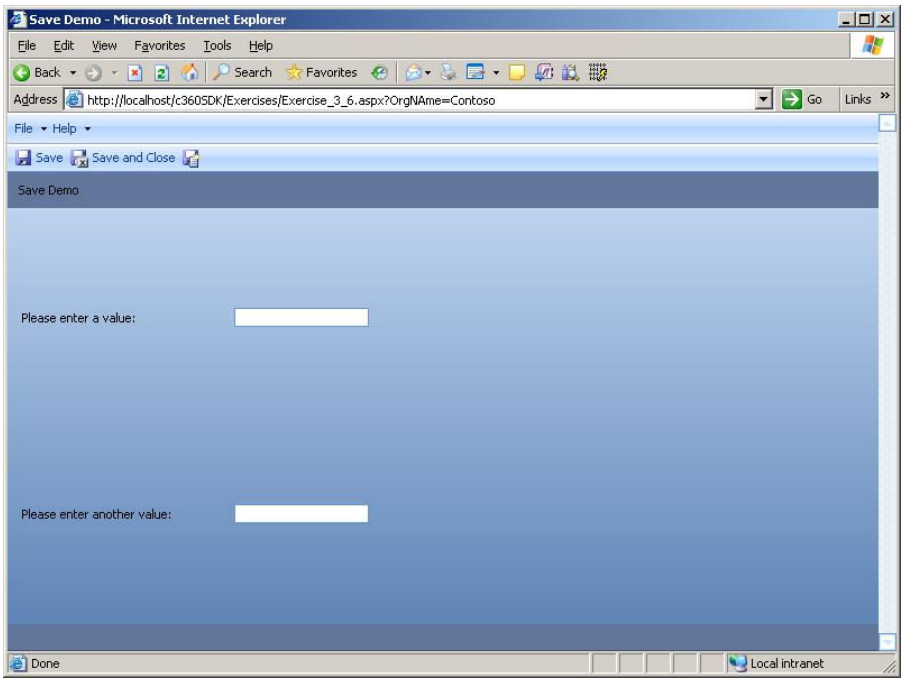
EditPage (cont'd)

Exercise 3.6

Build a page deriving from EditPage which:

- Has a file menu
- Has a button bar
- Has an ASP.NET web form control text field with the label "Please enter a value"
- Has a second ASP.NET web form control text field with the label "Please enter another value"
- Lays out the labels in a column spanning 25% of the screen and the textboxes in another column covering the remaining 75% of the screen
- Has a page title of "Save Demo"
- Validates the user input and makes sure that a value has been entered in both fields
- Gathers all user input
- Handles the server-side "save" event and saves the user input to a text file.

Exercise 3.6 Desired Output



EditPage (cont'd)

Exercise 3.6 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.EditPage" Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
	public override void BuildMenuArea() {
		// Define the Title of the EditPage
		Instance.PageTitle = "Save Demo";

		// Add validation and define save handler javascript functions
		Instance.ClientSideValidationHandler = "validate";
		Instance.ClientSidePreSaveHandler = "submit";

		// Continue to build the menu area
		base.BuildMenuArea();
	}

	public override void BuildContentArea() {
		// Create an HTML table which will allow us to layout the labels and text boxes in two columns
		HtmlTable layoutTable = new HtmlTable();
		layoutTable.Width = "100%";
		layoutTable.Height = "100%";

		// Add the first row to our table
		layoutTable.Rows.Add(new HtmlTableRow());

		// We need two columns: one for the label and one for the textbox
		layoutTable.Rows[0].Cells.Add(new HtmlTableCell());
		layoutTable.Rows[0].Cells.Add(new HtmlTableCell());

		// Make sure that the first column is 25% wide
		layoutTable.Rows[0].Cells[0].Width = "25%";

		// Add the label in the first cell of the first row
		layoutTable.Rows[0].Cells[0].Controls.Add(new System.Web.UI.LiteralControl("Please enter a value: "));

		// Add a text box in the second cell of the first row
		System.Web.UI.WebControls.TextBox textBox1 = new TextBox();
		textBox1.ID = "tb1";
		layoutTable.Rows[0].Cells[1].Controls.Add(textBox1);

		// Add the second row to our table
		layoutTable.Rows.Add(new HtmlTableRow());

		// Again, we need two columns: one for the label and one for the textbox
		layoutTable.Rows[1].Cells.Add(new HtmlTableCell());
		layoutTable.Rows[1].Cells.Add(new HtmlTableCell());

		// Add the label in the first cell of the second row
		layoutTable.Rows[1].Cells[0].Controls.Add(new System.Web.UI.LiteralControl("Please enter another value: "));
```

EditPage (cont'd)

Exercise 3.6 Solution (cont'd)

```
            // Add a text box in the second cell of the second row
            System.Web.UI.WebControls.TextBox textBox2 = new TextBox();
            textBox2.ID = "tb2";
            layoutTable.Rows[1].Cells[1].Controls.Add(textBox2);

            // Add our layout table to the page
            Instance.ContentArea.Controls.Add(layoutTable);
        }

        public override string SaveData(string stringToSave) {
            // Load the input string into a XML DOM object
            XmlDocument inputDocument = new XmlDocument();
            inputDocument.LoadXml(stringToSave);

            // Save the XML document to disk
            inputDocument.Save("C:\\Testing.xml");

            // Indicate success
            return "";
        }
</script>

<script language="javascript">
function validate(){
        if (document.all.tb1.value == "" || document.all.tb2.value == "") {
            alert("Please enter a value in both text boxes before saving.");
            return false;
        }
        return true;
}

function submit() {
        var stringToSave = "<stringToSave>"
        stringToSave += "<firstValue>" + document.all.tb1.value + "</firstValue>";
        stringToSave += "<secondValue>" + document.all.tb2.value + "</secondValue>";
        stringToSave += "</stringToSave>"
        return stringToSave;
}
</script>

<html>
        <head>
            <title>Exercise_3_6</title>
        </head>
        <body>
        </body>
</html>
```

GridDataPage

Overview

The GridDataPage class does not have any UI, and is used in conjunction with the Grid control in order to take control of the data retrieval process. This class allows you to fetch data from CRM, an external data source, CRM System Views, or CRM User Views, and output that data in an XML format for presentation in a Grid control.

While querying data outside of CRM, keep in mind that you are responsible for handling filtering, sorting, and paginating the data. The GridDataPage contains properties (such as PageNumber, JumpValue, FilterValue) that are passed to it from the Grid control that you can use in your queries to handle this functionality.

GridDataPage Properties

Property (type)	**Description/Comment**
AllRecords (bool)	Indicates if all of the records are displayed. `Instance.AllRecords = false;`
AutoQueryApi (string)	This property is used in conjunction with certain Microsoft CRM saved queries that expect you to specify an entity and a method at run time in order to get the data. However, since there is no user interface in Microsoft CRM (as of this writing) to allow a user to build one such query, this property is rarely used. An example of a saved query that expects this value is the "Contacts Associated View" and an example of a query API is "Contact.RetrieveByObject". `Instance.AutoQueryApi = "Contact.RetrieveByObject';`
DistinctRecords (bool)	Indicates whether you want the grid to filter duplicate records based on their unique identifier. This is particularly useful when fetching data using a FetchXml query that may return duplicate data since, as of this writing, the CRM query engine is unable to filter duplicates. `Instance.DistinctRecords = true;`
EntityId (string)	The GUID of the entity `Instance.EntityId = "{7044B4EC-E926-DB11-ADA4-00065BF12C53}";`
EntityTypeCode (int)	Used to specify the integer value representing the type of the "parent" record. This property is used when the grid is must display record related to a parent record such as when you list all contacts related to a parent account. `Instance.EntityTypeCode = 2;`

GridDataPage (cont'd)

GridDataPage Properties (cont'd)

Property (type)	Description/Comment
FetchXml (string)	The FetchXML string used to query CRM
FilterField (string)	The value of the field to filter the data on
FilterValue (string)	The value to filter the specified FilterField on.
IgnoreFilter (bool)	Indicates if you want to ignore the filters set in your query
JumpField (string)	Name of a field that will be used to specify a criteria. By default, this is used by the "Jump Bar" at the bottom of the grid.
JumpValue (string)	The value of the jumpfield passed when a user clicks on a letter
MoreRecords (bool)	Indicates if there are more records available
PageNumber (Int)	The data page number. You use this property if you want to display a page other than the first one by default.
RecordsPerPage (int)	Specifies the number of results you'd like per page on the grid.
ReturnedTypeCode (int)	The CRM type of the records displayed in the grid.
SortCol (String)	The name of the column to sort the data on by default.
SortDir (OrderType)	The direction of the sorting
ViewId (String)	CRM GUID of the view to use to get the data. Only necessary if displaying a CRM System, or User View.
ViewType (ViewTypes)	CRM ViewType of the view to use to get the data Two possible choices: • SystemQuery • UserQuery Only necessary if displaying a CRM System, or User View

GridDataPage (cont'd)

GridDataPage Methods

Method (return)	Description/Comment
AddDataTableToReturn (Void)	Allows you to return a datatable as the source of a grid control.
AddResultSetToReturn (Void)	Adds a resultset string to be returned to the grid. Calling this method several times allows you to combine data from multiple sources.
FilterResultSet (String)	Filters the resultset in XML after fetching. (Note: for performance reasons, it is recommended to filter data inside the fetchXML)
GetLookupTypes (String)	Returns a XML string containing the list of entity types that are valid for a given lookup field.
GetPickListValues (String)	Returns a XML string containing the list of available value for a given picklist field.
GetSavedQueryLayout (String)	Returns a XML string containing information about a saved query such as the list of configured columns, sorting direction, etc. Invoked by the grid in order to refresh the display when the user selects a different view from the views selector.
GetStateValues (String)	Returns a XML string containing the list of available values for the state field.
GetStatusValues (String)	Returns a XML string containing the list of available values for the status field.

GridDataPage Events

Event (return)	Description/Comment
ExportToMedia (Void)	Overridable method to export data to a custom media (e.g. PDF, custom Excel doc)
FetchData (Void)	Overridable method to implement custom data fetching scheme

GridDataPage (cont'd)

Resultset format

The AddResultSetToReturn method accepts a XML string containing the data to be presented in the grid. The format is designed to give you total control on how you want your data to be presented to the user. For example, it allows you to specify the tooltip of each cell (by default the tooltip is the text in the cell), you can specify a "sort value" which will determine how a row will be sorted, etc. This format is compatible with the Resultset xml you would get from the Microsoft CRM API if you execute a query using fetchXml but you can decorate certain nodes with additional attributes to control how your data is presented.

Here are the rules to follow in order to produce a valid Resultset XML string:

1. A resultset XML string always begin with a `<resultset>` tag and always end with a `</resultset>` tag.
2. The "resultset" node can contain an infinite number of "result" node which represent one record
3. Each "results" node must have a "objectType" attribute containing the integer value representing the type of the record. For example "1" represents an account, "2" represents a contact and "10000" represents a custom entity. Please note that the c360 grid supports resultsets with many different objectTypes. E.g.: `<result objectType="2">`
4. Each "result" node must have a "primaryIdField" attribute which indicates the name of the primary key field. For example, in the case of an account, the primaryField would be "accountid". E.g.: `<result primaryIdField="contactid">`
5. Each "result" node can have an infinite number of nodes named after the field they represent. E.g.: `<name>John Smith</name>`.
6. The content of each node is the actual data
7. Each node inside the "result" can have a "sortvalue" attribute which will be used to determine the position of this record when the data is sorted by the field in question. If no "sortvalue" is specified, the content of the node will be used. We recommend keeping this value to a reasonable length (no more than 20 characters) to limit the size of the resulting XML string. E.g.: `<name sortvalue="Smith, John">John Smith</name>`.
8. Each node inside the "result" can have a "formattedvalue" attribute which will be displayed in the grid. E.g.: `<balance sortvalue="000008" formattedvalue="$8.00">8</balance>`.
9. Each node inside the "result" can have a "name" attribute which serves the exact same purpose as the "formattedvalue" attribute. E.g.: `<accounttype name="Premium Customer">1</accounttype>`
10. If a node has both a "name" and a "formattedvalue", the "name" takes precedence.

GridDataPage (cont'd)

Resultset format (con't)

11. Each node inside the "result" can have a "date" and/or "time" attributes to represent date fields. For example: `<createdon date="December 31 2005" time="midnight">2005/12/31T00:00:00</createdon>`. The end user will be presented with the combination of these two attributes: "December 31 2005, midnight".
12. Each node inside the "result" can have any of the following attributes if the field is displayed as an 'ExternalLink'. If the field is not displayed as an 'ExternalLink', these attributes will be ignored:
    a. "url": the destination of the external link
    b. "width": the height of the popup window
    c. "height": the width of the popup window.
    d. "customParams": custom parameters to control the style of the popup window such as "status=0" to hide the status bar.

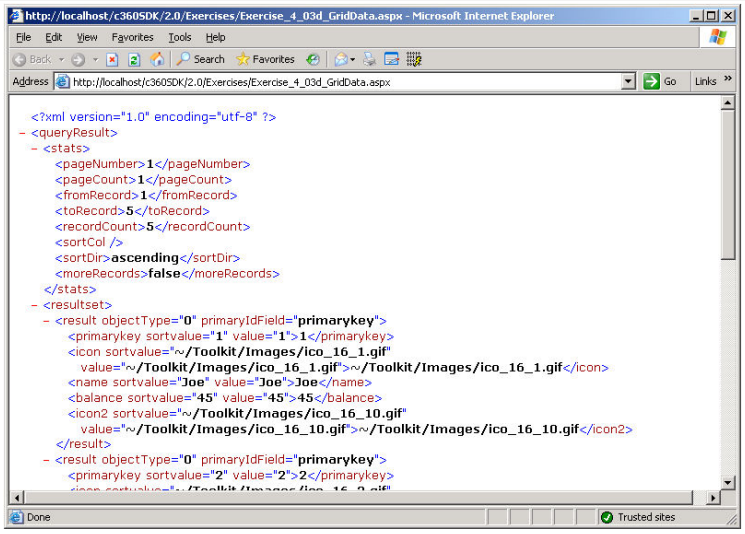
    E.g.: `<website url=http://www.microsoft.com" width="800" height="600">Microsoft</website>`
13. Each node inside the "result" can have a "tooltip" attribute which will be displayed when the user moves his mouse over the cell. If the attribute is missing, the entire content of the cell is used as the tooltip. Please note that Internet Explorer enforces a length limit to tooltips and therefore your tooltip may be truncated.

GridDataPage (cont'd)

GridDataPage Sample

The following sample illustrates how to create a GridDataPage and the corresponding XML output from a query to pull all contacts and accounts that's names begin with the letter "A".

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.GridDataPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.WebServices" %>
<%@ Import Namespace="System.Data" %>

<script runat="server">
public void FetchData() {
        // Retrieve all accounts starting with the letter "A"
        string fetch = "<fetch mapping='logical'><entity name='account'><all-attributes/><filter type='and'>";
        fetch += "<condition attribute='name' operator='like' value='A%'/>";
        fetch += "</filter></entity></fetch>";
        String strResultSet =CrmSservice.ExecuteFetchXml(fetch);

        //map the account field "name" to "fullname" so that all of the results
        //(both accounts and contacts) fall under the same column.
        strResultSet = strResultSet.Replace("<name>", "<fullname>");
        strResultSet = strResultSet.Replace("<name/>", "<fullname/>");
        strResultSet = strResultSet.Replace("<name ", "<fullname ");
        strResultSet = strResultSet.Replace("</name>", "</fullname>");

        // Retrieve all contacts having the lastname that starts with the letter "A"
        string fetch2 = "<fetch mapping='logical'><entity name='contact'><all-attributes/><filter type='and'>";
        fetch2 += "<condition attribute='lastname' operator='like' value='A%'/>";
        fetch2 += "</filter></entity></fetch>";
        String strResultSet2 = CrmService.FetchXml(fetch2);

        //GridDataPage method to add the results from the fetch into an XML formatted return
        AddResultSetToReturn(strResultSet, 1, "accountid");
        AddResultSetToReturn(strResultSet2, 2, "contactid");
}
</script>
```



SavePage

Overview

The SavePage class is used to build a page that will be presented in a dialog window and where the user can click on the “Ok” or “Cancel” button to save data. The user can also be prompted for additional data prior to executing the save as well.

SavePage Properties

Property (type)	**Description/Comment**
ClientSidePreSaveHandler (string)	Specifies the name of a JavaScript method that will be called after the data has been successfully validated. The string returned by this client-side method is automatically passed to the SaveData server-side method.
ClientSideSaveErrorHandler (string)	Specifies the name of a JavaScript method that will be called if there is an error saving the data. Allows you to display a custom error message, or perform additional logic.
ClientSideSaveSuccessHandler (string)	Specifies the name of a JavaScript method that will be called if the save operation succeeds. Allows you to display a custom message or perform additional logic.
ClientSideValidationHandler (string)	Specifies the name of a JavaScript method that will be called when the data on the screen needs to be validated. The JavaScript must return true to proceed with saving the data, or false to stop the processing.

SavePage Methods

Method (return)	**Description/Comment**
AddCancelButton (void)	Adds a “Cancel” button to the bottom section of the window. The caption is automatically added and also translated in the appropriate language. The window is automatically closed when the user clicks on this button. `Instance.AddCancelButton();`
AddOkButton (void)	Adds a “Ok” button to the bottom section of the window. The caption is automatically added and also translated in the appropriate language. The JavaScript specified in the “ClientSidePreSaveHandler” will be executed when the users clicks on this button. `Instance.AddOkButton();`

SavePage (cont'd)

EditPage Event

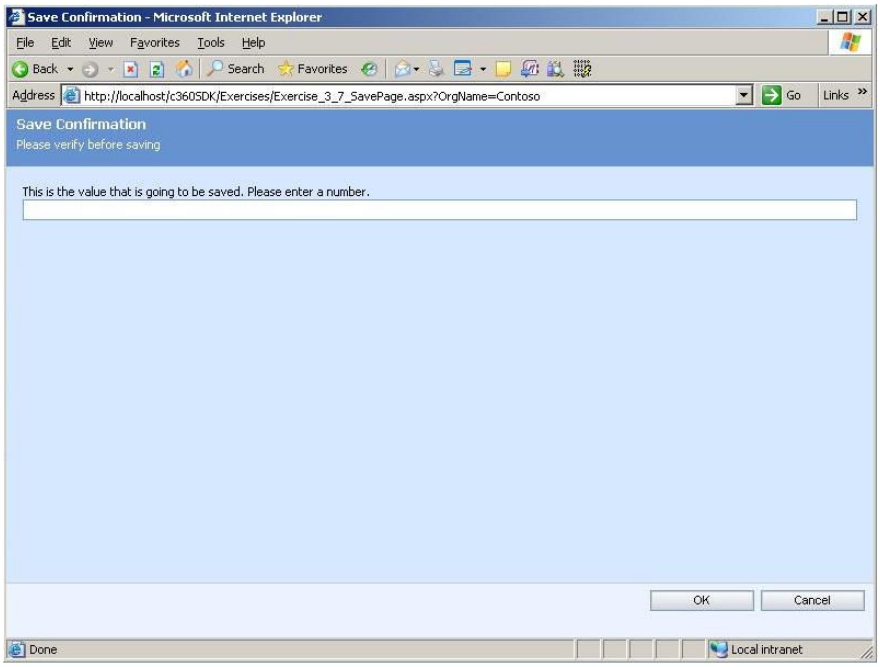
Method (return)	**Description/Comment**
SaveData (string)	This event is raised on the server-side once for every selected record after the user clicks on the "Ok" button. This event receives one string parameter containing the value returned by the client-side save event handler. This method can return a string containing any error messages or an empty string when it completes successfully.

Exercise 3.7

Create a page deriving from SavePage which:

- Has a dialog title of "Save Confirmation"
- Has a dialog description of "Please verify before saving"
- Contains the text "This is the value that is going to be saved" in the content area.
- Contains a text box with a pre-filled value+
- Has an "OK" and a "Cancel" button
- Contains a ClientSidePreSaveHandler client script to create an XML string that will be passed to the SaveData method

Exercise 3.7 Desired Output



SavePage (cont'd)

Exercise 3.7 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.Dialogs.SavePage" %>
<script runat="server">
	public override void BuildMenuArea() {
		Instance.ClientSidePreSaveHandler = "preSave()";
		Instance.DialogTitle = "Save Confirmation";
		Instance.DialogDescription = "Please verify before saving";
		base.BuildMenuArea();
	}

	public override void BuildContentArea() {
		Instance.ContentArea.Controls.Add(new System.Web.UI.LiteralControl("This is the value that is going to be saved"));
		System.Web.UI.WebControls.TextBox textBox1 = new TextBox();
		textBox1.ID = "input";
		textBox1.Text = "sample value";
		Instance.ContentArea.Controls.Add(textBox1);
	}

	public override void BuildFooterArea() {
		Instance.AddOkButton("btnOk");
		Instance.AddCancelButton("btnCancel");
	}

	public override string SaveData(string stringToSave) {
		return stringToSave;
	}
</script>

<script language="javascript">
function preSave() {
	var input = document.all.input.value;
	var xml = "<xml><object>" + input + "</object></xml>";
	return xml;
}
</script>
```

TreeViewDynamicContent

Overview

The TreeViewDynamicContent class does not have any UI, and is used in conjunction with the TreeView control in order to take control of the data retrieval process. Pages extending from this class should query data and dynamically generate and XML file, to be used to dynamically generate content when a node is clicked in a TreeView control.

TreeViewDynamicContent Methods

Method (return)	**Description/Comment**
AddTreeViewItemToReturn (Void)	Add a TreeViewItem that will later be returned to the TreeView control

TreeViewDynamicContent Events

Event (return)	**Description/Comment**
FetchData (Void)	Overridable method to implement custom data fetching scheme

TreeViewDynamicContent (cont'd)

TreeViewDynamicContent Sample

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.TreeViewDynamicContent" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>
<%@ Import Namespace="c360.Toolkit.CRM " %>

<script runat="server">
	public override void FetchData() {
		int thisMonth = System.Int32.Parse(base.ObjId); //for using this page with a treeview control
		//int thisMonth = 10; //for testing contentpage directly
		string beginDate = thisMonth.ToString() + "/01/2005";
		int endDay = DateTime.DaysInMonth(2005,thisMonth);
		string endDate = thisMonth.ToString() + "/" + endDay.ToString() + "/2005";

		ConditionExpression objConditions = new ConditionExpression();
		objConditions.AttributeName = "birthdate";
		objConditions.Operator = ConditionOperator.Between;
		objConditions.Values = new object[]{ beginDate, endDate };

		// Define what we need to fetch
		FilterExpression objFilter = new FilterExpression();
		objFilter.FilterOperator = LogicalOperator.And;
		objFilter.Conditions = new ConditionExpression[] {objConditions};

		// Get the contact
		QueryExpression objQuery = new QueryExpression();
		objQuery.EntityName = EntityName.contact.ToString();
		objQuery.ColumnSet = new AllColumns();
		objQuery.Criteria = objFilter;

		RetrieveMultipleRequest objRequest = new RetrieveMultipleRequest();
		objRequest.Query = objQuery;

		RetrieveMultipleResponse objContacts = (RetrieveMultipleResponse) CrmService.Execute(objRequest);

		for(int i=0;i<objContacts.BusinessEntityCollection.BusinessEntities.Length;i++)
		{
			contact thisContact = (contact) objContacts.BusinessEntityCollection.BusinessEntities[i];
			string name = ((contact)objContacts.BusinessEntityCollection.BusinessEntities[i]).fullname;
			string contactid = ((contact)objContacts.BusinessEntityCollection.BusinessEntities[i]).contactid.ToString();
			AddTreeViewItemToReturn(name, 2, contactid, null, false, false, false);
		}
	}
</script>
```

4. Using Visual Controls

Controls Overview

Purpose / Background

Visual controls available through the c360 SDK allow developers to quickly and easily create repeatable controls that take end-user input. These controls have properties and methods (both server-side and client-side) that developers will use to communicate with them.

These controls will be the foundation of your custom CRM development, and two very important things to remember for these controls are:

1. **All controls are accessed through interfaces.**
2. **All controls are created using the ControlFactory class.**

Available Controls in the c360 SDK

There are currently 13 controls available in the c360 SDK, with more controls to come with future versions of the SDK. The following is a list of these controls that we will detail during this training:

- DatePicker
- DualList
- Grid
- GridPreferences
- LeftNavBar
- List
- Lookup
- Menu
- MultiPicklist
- PickList
- Radio
- TabBar
- TreeView

Usage

To create control, you use the control factory and to add them to the page, you need to add them to the Controls list on the page instance. Here is an example:

```
Using c360.Toolkit.SDK.UI;

public override void BuildContentArea()
{
        IDatePicker myDatePicker = ControlFactory.CreateDatePicker();
        myDatePicker.ID = "birthDay";
        Instance.ContentArea.Controls.Add(myDatePicker.Control);
}
```

DatePicker

Overview

The DatePicker is a Calendar-like date selector.

DatePicker Server Side Properties

Server Side Property (type)	Description/Comments
ID (string)	It is very important that you set this property to a unique value. `myDatePicker.ID = "birthdayDatePicker";`
Disabled (bool)	Allows you to disable or enable the control `myDatePicker.Disabled = false;`

SelectedDate (CrmDateTime)	Allows you to set the initial date selected in the calendar.
	`CrmDateTime birthday = newCrmDateTime();`
	`birthday.Value = "07/10/1973";`
	`myDatePicker.SelectedDate = birthday;`

OnChangeEventHandler (string)	This property allows you to specify the name of a JavaScript function that will be called when the user changes the selection. `myDatePicker.OnChangeEventHandler = "jsDateChangeHandler";`
UTCFormatDate (string)	Allows you to set the Date in UTC format.

DatePicker Server Side Methods

[There are no SDK-specific methods for this control]

DatePicker Client Side Properties

Client Side Property (type)	Description/Comments
value (string)	The date selected by the user expressed in the user's time zone and date format
returnValue (string)	The date selected by the user expressed in UTC format. This is the value you must use if you intend to save the user selected date in the CRM database because all CRM dates are saved in UTC format.

DatePicker Client Side Methods

[There are no SDK-specific methods for this control]

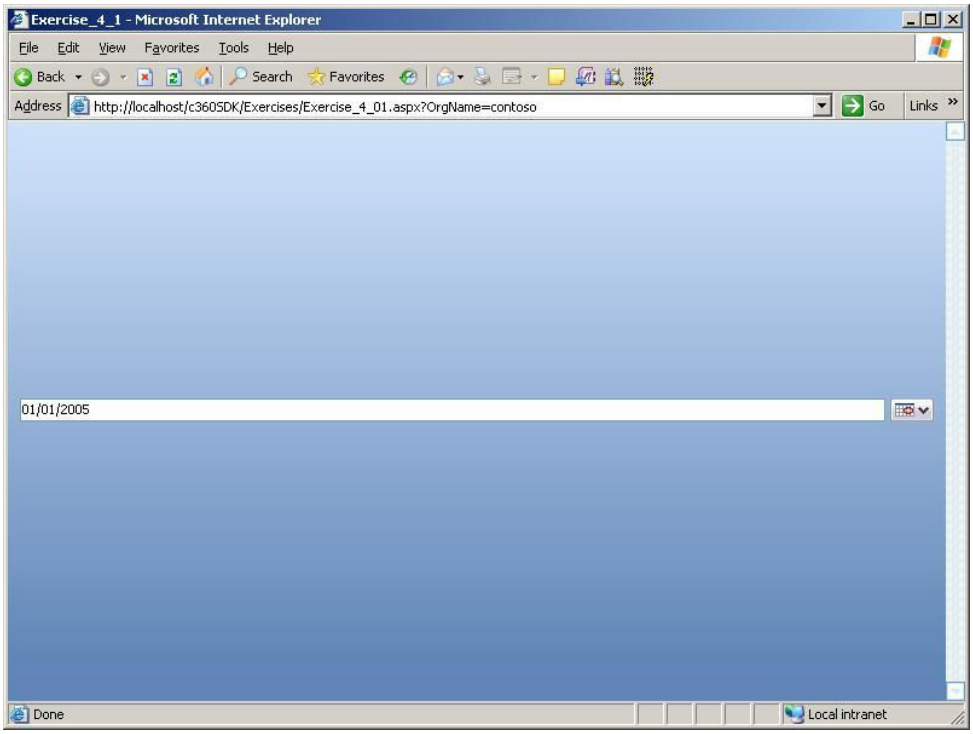
DatePicker (cont'd)

Exercise 4.1: DatePicker

Build a page deriving from Area which:

- Displays a DatePicker with a default date of January 1, 2005

Exercise 4.1 Desired Output



Exercise 4.1 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildMenuArea() {
                IDatePicker objDatePicker = ControlFactory.CreateDatePicker();
                objDatePicker.ID = "TestDatePicker";
                c360.Toolkit.CRM.CrmDateTime beginDate = new c360.Toolkit.CRM.CrmDateTime();
                beginDate.Value = "01/01/2005";
                objDatePicker.SelectedDate = beginDate;
                Instance.ContentArea.Controls.Add(objDatePicker.Control);
        }
</script>
```

DualList

Overview

The DualList consists of two side-by-side lists controls and allows the end user to highlight one or more items from a list and "send" his selection to the other list. Four buttons are available in between the two lists to allow the user to transfer items from one list to the other. By convention, the list on the left contains a list of items to choose from and the list on the right represents the items selected by the user.

Dualist Server Side Properties

Server Side Property (type)	**Description/Comments**
AllowMoveAllLeft (bool)	Indicates if the user can move all items from the list on the right to the one on the left at once. If true, a button will be available between the two lists to allow to user to perform this action. `myDualList.AllowMoveAllLeft = false;`
AllowMoveAllRight (bool)	Indicates if the user can move all items from the list on the left to the one on the right at once. If true, a button will be available between the two lists to allow to user to perform this action. `myDualList.AllowMoveAllRight = false;`
AllowMoveLeft (bool)	Indicates if the user can move the selected item(s) from the list on the right to the one on the left. If true, a button will be available between the two lists to allow to user to perform this action. `myDualList.AllowMoveLeft = true;`
AllowMoveRight (bool)	Indicates if the user can move the selected item(s) from the list on the left to one on the right. If true, a button will be available between the two lists to allow to user to perform this action. `myDualList.AllowMoveRight = true;`
Height (Unit)	The height of the rendered Dual list. `myDualList.Height = Unit.Percentage(100);`
ID (string)	It is very important that you set this property to a unique value. `myDualList.ID = "favoriteSportsPicker";`

DualList (cont'd)

Dualist Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
LeftList (IList)	Reference to the left list. Allows you to add items to this list. See the "List" control for more details. `IList myLeftList = myDualList.LeftList;`
RightList (IList)	Reference to the right list. Allows you to add items to this list. See the "List" control for more details. `IList myRightList = myDualList.RightList;`
TransferItemEventHandler (string)	This property allows you to specify the name of a JavaScript function that will be called when the user changes the selection. `myDualList.TransferItemEventHandler = "jsTransferItemHandler";`
Width (Unit)	The width of the rendered Dual list. `myDualList.Width = Unit.Percentage(100);`

DualList Server Side Methods

[There are no SDK-specific methods for this control]

DualList Client Side Properties

NOTE: on the client side, there is no dual list control per-se but rather two lists, one called "<unique ID>_LeftList" and the other called "<unique ID>_RightList".

Here is an example showing how you can count the number of items in a list.

```
Alert("The number of item in the left list is: " + myDualList_LeftList.GetAllValues().length);
```

For more details on the client-side properties of those two lists, please see the "List" control.

DualList (cont'd)

DualList Client Side Methods

Client Side Methods (return)	**Description/Comments**
MoveAllItemsFromLeftToRight_<unique ID> (string)	Moves all the items in the left list to the right list and remove them from the left list. `MoveAllItemsFromLeftToRight_myDualList();`
MoveAllItemsFromRightToLeft_<unique ID> (string)	Moves all the selected items in the right list to the left list and remove them from the right list. `MoveAllItemsFromRightToLeft_myDualList();`
MoveItemsFromLeftToRight_<unique ID> (string)	Moves the selected item(s) in the left list to the right list and remove them from the left list. `MoveItemsFromLeftToRight_myDualList();`
MoveItemsFromRightToLeft_<unique ID> (string)	Moves the selected item(s) in the right list to the left list and remove them from the right list. `MoveItemsFromRightToLeft_myDualList();`

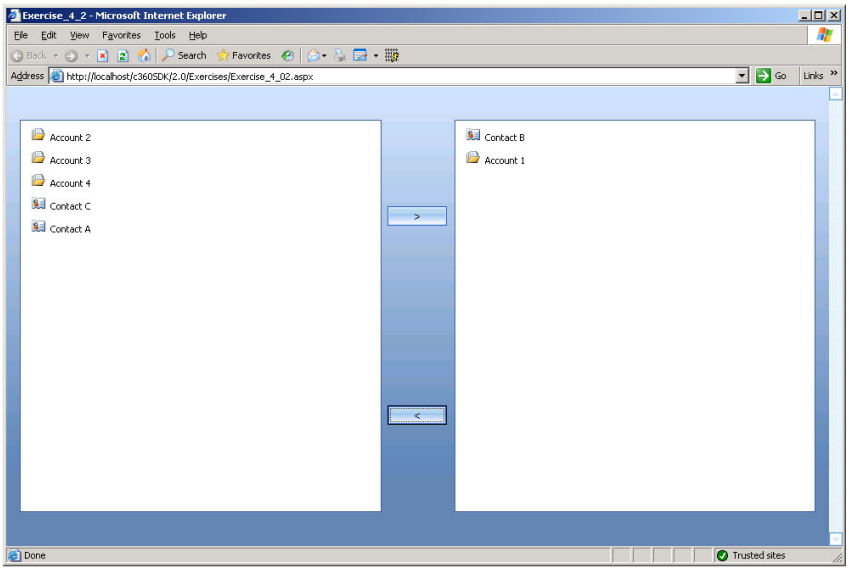
DualList (cont'd)

Exercise 4.2: DualList

Create a page deriving from Area which:

- The content area contains a dual list containing:
  - 4 sample accounts listed on the left
  - 3 sample contacts listed on the right
- The third account on the list is pre-selected
- The "<<" and ">>" (move all left / right) buttons not visible

Exercise 4.2 Desired Output



Exercise 4.2 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildContentArea() {
                IDualList objDualList = ControlFactory.CreateDualList();
                objDualList.Control.ID = "MyDualList";
                objDualList.LeftList.Items.Add(ControlFactory.CreateListItem("Account 1", 1, "Account 1", false));
                objDualList.LeftList.Items.Add(ControlFactory.CreateListItem("Account 2", 1, "Account 2", false));
                objDualList.LeftList.Items.Add(ControlFactory.CreateListItem("Account 3", 1, "Account 3", true));
                objDualList.LeftList.Items.Add(ControlFactory.CreateListItem("Account 4", 1, "Account 4", false));
                objDualList.RightList.Items.Add(ControlFactory.CreateListItem("Contact A", 2, "Contact A", false));
                objDualList.RightList.Items.Add(ControlFactory.CreateListItem("Contact B", 2, "Contact B", false));
                objDualList.RightList.Items.Add(ControlFactory.CreateListItem("Contact C", 2, "Contact C", false));
                objDualList.AllowMoveAllLeft = false;
                objDualList.AllowMoveAllRight = false;
                objDualList.AllowMoveLeft = true;
                objDualList.AllowMoveRight = true;
                Instance.ContentArea.Controls.Add(objDualList.Control);
        }
</script>
```

Grid

Overview

The Grid control is the most powerful and complex control. It allows developers to present CRM data in a grid format that looks and acts like the Microsoft CRM grid. This grid will automatically filter, sort, and page the data, as well as being able to display a standard search criteria at the top to mimic the search box at the top of Microsoft's grid except that our search is configurable and can be set to filter on any field and is not limited to one field. The grid supports three modes of data fetching including specifying a fetchXML query, specifying the GUID of an existing saved query, and custom data fetching where the developer fetches the data from whatever data source they'd like (e.g. external Sql data).

The grid was designed to be able to display multiple entity types at the same time contrary to Microsoft's grid which can only present one type at a time. Some other highlights of the grid control include:

- Filtering the result based on user input
- CRM views can be used to define what data should be fetched and which columns should be displayed
- Grid includes menu bar with standard Microsoft CRM actions
- Developer can add custom actions
- Data can be printed and exported to Excel or to custom media
- Grid can react to input such as click, double-click, right-click

How does the Grid get the data?

The grid was designed so that it is very easy for the developer to specify what must be fetched and displayed without worrying about how to get the data. At the same time, the grid is flexible enough to allow the developer to override the default behavior and take control of the fetching process.

There are 3 different ways the grid can get the data:

1. You can specify the system id of an existing CRM saved query and the grid will take care of figuring out what columns must be displayed and what data must be fetched. It will also take care of fetching the data and presenting it in the grid. This is the easiest way to present CRM data in the grid.
2. You can specify the FetchXml necessary to get the data and the grid will take care of getting the data and displaying it. In this situation, you are responsible for indicating the columns you want to be displayed.
3. You can take control of the entire process and fetch the data yourself. In this situation, you are responsible for specifying the columns, fetching the data and converting it to the format expected by the grid. This is the perfect solution to allow you to extract data from an external system and display it in a grid that looks like the Microsoft CRM grid.

Grid (cont'd)

The grid is in fact a combination of a control and a page called the "GridData" page. The control renders as HTML on a main page and presents a familiar look to the end user while the page is responsible for fetching the data in an asynchronous manner. The SDK comes with a standard GridData page and the "FetchDataPage" property on the grid control allows you to specify an alternate web page where you take control of the data fetching process. The SDK includes a page class called "GridData" that your custom page must derive from and which makes it easier for you to send the data back to the grid.

How is the data filtered and sorted?

If you use the grid's built-in ability to query data by either using FetchXml or a saved query, the data will be automatically filtered, "paged" and sorted for you. For this to happen, the filtering information must be specified in the "FilterField" and "FilterValue" properties and the sorting information must be provided in the "SortField" and the "SortDir" fields. Please note that the default value for "FilterField" is the "description" field of the entity currently displayed in the grid. In addition to filtering your data, the grid will also "page" the result which means that only 25, 50, 75 or 100 records will be displayed at a time, based on the user preference.

If you decide to take control of the fetching process, you also benefit from these features: the SDK's GridData page has all the built-in logic to filter, sort and page your data. This means, for example, that your custom GridData page could retrieve all the records from an external data source but the user would only see 50 at a time in the order he requested without you having to write an additional line of code. However, you must consider that this is not necessarily the most efficient way to query and present your data. Therefore, you should implement your own logic to add the filtering criteria to your query and only retrieve appropriate records in the appropriate order.

I want to take control of the fetching process. How do I pass the data back to the grid?

When you implement a custom fetching process, you create a new ASPX page and derive from the GridData page. In your new page, you override the "FetchData" method and you retrieve your data. The last obstacle is to return the data to the user's browser to be displayed. The GridData page provides two ways to do this:

- AddDataTableToReturn: this method accepts a DataTable as a parameter. The rows in the datatable will be displayed as rows in the grid. This method is very convenient if you use ADO.NET to get your data from an external data source but it offers very little flexibility to format the result.
- AddResultsetToReturn: this method accepts a XML string with a very flexible format that allows you to precisely format your data.

The resultset format is discussed in more details in chapter 3.

Grid (cont'd)

Grid Server Side Properties

Server Side Property (type)	Description/Comments
Actions (MenuItemsCollection)	Collection of actions that will be added to the standard "Actions" menu. You can add to this collection by specifying the caption of the menu item, the path to the icon to be displayed in front of the caption (or null if you don't want any icon) and the JavaScript to be executed when the user clicks on this menu item. This gives you the flexibility of executing any JavaScript you want when an item is selected. If you plan on writing a custom action based on the "Action" page class provided by the SDK, you should use the "CustomActions" collection instead. `objMyGrid.Actions.Add(new MenuItem("Test", "/Images/Testing.gif", "alert('Test');"));`
AllowCreateNew (bool)	Indicates if the "Create New" button is to be displayed on the menu bar above the grid. Please note that the button will not be displayed if the user doesn't have the appropriate CRM permission, even when this property is set to true. `objMyGrid.AllowCreateNew = true;`
AllowExportToExcel (bool)	Indicates if the "Export to Excel" button is to be displayed on the menu bar above the grid. Please note that the button will not be displayed if the user doesn't have the appropriate CRM permission, even when this property is set to true. `objMyGrid.AllowExportToExcel = true;`
AllowMultiSelect (bool)	Indicates whether the user can highlight more than one record of not. `objMyGrid.AllowMultiSelect = false;`

Grid (cont'd)

Grid Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
AllowPrint (bool)	Indicates if the "Print" button is to be displayed on the menu bar above the grid. Please note that the button will not be displayed if the user doesn't have the appropriate CRM permission, even when this property is set to true. `objMyGrid.AllowPrint = true;`
AllowPrintReports (bool)	Indicates if the "Print" button is to be displayed on the menu bar above the grid. `objMyGrid.AllowPrintReports= true;`
AutoQueryAPI (string)	This property is used in conjunction with certain Microsoft CRM saved queries that expect you to specify an entity and a method at run time in order to get the data. However, since there is no user interface in Microsoft CRM (as of this writing) to allow a user to build one such query, this property is rarely used. An example of a saved query that expects this value is the "Contacts Associated View" and an example of a query API is "Contact.RetrieveByObject". `objMyGrid.AutoQueryApi = "Contact.RetrieveByObject";`
BackGroundColor (Color)	You use this property to specify an alternate background color if you do not want the default white background. `objMyGrid.BackGroundColor = Color.Black;`
ClickEventHandler (string)	Name of a JavaScript function that will be invoked when the user clicks on a row in the grid. This function will be passed 3 parameters: the CRM type of the record, the unique identifier of the row and also an xml node containing all the data about this record that was fetched by the GridData page. `objMyGrid.ClickEventHandler = "jsClickHandler";`
ColumnHeaders (ColumnHeaderCollection)	This property allows you to specify the fields you want to display to the user. You can omit this property if you have specified a value in the "ViewId" property since the columns information is part of the saved query definition and will be automatically retrieved from you. For each column you want to display, you must specify the caption of the header, the name of the xml node where the data can be found and the default width in pixels. You can optionally specify the type of column. `objMyGrid.ColumnHeaders.Add(new ColumnHeader("Birthday ", "dateofbirth", 200));`

Grid (cont'd)

Grid Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
CustomActions (CustomGridActionsCollection)	Collection of custom actions based on the "Action" SDK page that will be added to the standard "Actions" menu. You can add to this collection by specifying the caption of the menu item, the URL of the action page, the width and height of the dialog that will show the action page. You can optionally specify a minimum and a maximum number of items that must be selected in the grid by the user before your action can be executed. If you do not intend to write your custom action based on the "Action" page class provided by the SDK, you should use the "Actions" collection instead. `objMyGrid.CustomActions.Add(new ControlFactory.CreateCustomGridAction("Action 1", "/Actions/Action1.aspx", 200, 300, 3, 4));`
DistinctRecords (bool)	Indicates whether you want the grid to filter duplicate records based on their unique identifier. `objMyGrid.DistinctRecords = true;`
DoubleClickEventHandler (string)	Name of a JavaScript function that will be invoked when the user double-clicks on a row in the grid. See the "ClickEventHandler" property for more details. `objMyGrid.DoubleClickEventHandler = "jsDoubleClickHandler";`
EnableEntityLink (bool)	This property needs to be set to true if you are using any columns of type EntityLink. This property is false by default, but when set to true will allow links to lookup records when specified as the column type. `objMyGrid.EnableEntityLink = true;`
EntityId (string)	Used to specify the unique identifier of the "parent" record. Just like the AutoQueryAPI property, this property is rarely used because it is used only when the grid is using certain Microsoft CRM queries that are designed to display data related to a "parent" record. `objMyGrid.EntityId = "A1D14F8C-24E5-4C5B-939B-EA864CF0DFE9";`
EntityTypeCode (int)	Used to specify the integer value representing the type of the "parent" record. Just like the AutoQueryAPI property, this property is rarely used because it is used only when the grid is using certain Microsoft CRM queries that are designed to display data related to a "parent" record. `objMyGrid.EntityTypeCode = 112;`

Grid (cont'd)

Grid Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
FetchDataPage (string)	URL to a custom page that will be responsible for fetching the data. This page MUST derive from the "GridData" page available in the SDK. `objMyGrid.FetchDataPage = "/Samples/MyCustomGridDataPage.aspx";`
FetchXml (string)	FetchXml query that specifies what data is to be fetched. `objMyGrid.FetchXml = "<fetch mapping='logical'><entity name='contact'><all-attributes/><filter type='and'><condition attribute='fullname' operator='like' value='t%'/></filter></entity></fetch>";`
FilterField (string)	Name of a field that will be used to specify a criteria. By default, this is used by the "Search" text box at the top of the grid. `objMyGrid.FilterField = "address1_stateorprovince";`
FilterValue (string)	Value that will be used to specify a criterion. `objMyGrid.FilterValue = "NY";`
ID (string)	It is very important that you set this property to a unique value. `objMyGrid.ID = "gridNewYorkCustomers";`
IgnoreFilter (bool)	Indicates if you want to ignore the filters set in your query. `objMyGrid.IgnoreFilter = false;`
IncludeStandardActions (bool)	Indicates whether the standard Microsoft CRM actions pertinent to the current entity are to be included in the "Actions" menu displayed above the grid. `objMyGrid.IncludeStandardActions = true;`
JumpField (string)	Name of a field that will be used to specify a "jump" criteria. This is used by the "Jump Bar" at the bottom of the grid. `objMyGrid.JumpField = "firstname";`

Grid Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
JumpValue (string)	Value that will be used to specify a "jump" criteria. `objMyGrid.JumpValue = "A";`
MenuItems (IMenuItemsCollection)	Collection of sub-menus that will be rendered on the menu bar above the grid. Sub-Menus that do not have child items will be rendered as buttons and those with child items will be rendered as pull-down menus. You can add to this collection by specifying the title of the menu item, the path to the icon to be displayed next to the title, the tooltip to be displayed to the user when he moves his mouse over the button, and the JavaScript statement to be executed when the button is clicked. See the "Menu" controls for more details and examples.
PageNumber (int)	The data page number. You use this property if you want to display a page other than the first one by default. `objMyGrid.PageNumber = 2;`
PagingStyle (PagingStyles)	Allows you to specify the style of the paging indicators below the grid. The two possible values are: "None" and "Standard". `objMyGrid.PagingStyle = PaginStyles.Standard;`
PostRefreshEventHandler (string)	Name of a JavaScript function to be called after the data presented in the grid has been refreshed. No parameters are passed to this function. `objMyGrid.PostRefreshEventHandler = "jsGridPostRefreshHandler";`
PreSelectedIds (ArrayList)	Collection of unique identifiers that you want to be initially selected if they are present in the grid.
RecordsPerPage (int)	The number of records to be presented on each page. By default, the grid will respect the value set by the user in his Microsoft CRM options but this property allows you to override this default value. `objMyGrid.RecordsPerPage = 1000;`
RefreshEventHandler (string)	Name of a JavaScript function to be called before the data presented in the grid is refreshed. No parameters are passed to this function. `objMyGrid.RefreshEventHandler = "jsGridPreRefreshHandler";`

Grid (cont'd)

Grid Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
ReturnedTypeCode (int)	The CRM type of the records displayed in the grid. `objMyGrid.ReturnedTypeCode = 2;`
RightClickEventHandler (string)	Name of a JavaScript function that will be invoked when the user right-clicks on a row in the grid. See "ClickEventHandler" for more details. `objMyGrid.RightClickEventHandler = "jsRightClickHandler";`
SearchingStyle (SearchingStyles)	Allows you to specify the style of the searching area displayed above the grid. The three possible values are: "None", "Standard" and "Quick". `objMyGrid.SearchingStyle =SearchingStyles.None;`
ShowBorder (bool)	Indicates if you want to display a border around the grid or not. `objMyGrid.ShowBorder = true;`
ShowEntityIcon (bool)	Indicates if you want to display an icon next to each record to represent their CRM type or not. `objMyGrid.ShowEntityIcon = true;`
ShowJumpBar (bool)	Indicates if you want to display the "Jump Bar" at the bottom of the grid or not. `objMyGrid.ShowJumpBar = true;`
ShowMenu (bool)	Indicates if you want to display the menu bar above the grid or not. `objMyGrid.ShowMenu = true;`
SortCol (string)	Name of the column by which the data is to be sorted by default. `objMyGrid.SortCol = "lastname";`
SortDir (OrderType)	Indicates the sorting direction. The two possible values are "Ascending" and "Descending". `objMyGrid.SortDir = OrderType.Ascending;`

Grid (cont'd)

Grid Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
ViewId (string)	Unique identifier of the Saved Query to be used to fetch the data. `objMyGrid.ViewId = "00000000-0000-0000-00AA-000010001001";`
ViewsSelectorStyle (ViewsSelectorStyles)	Allows you to specify the style of the views selector displayed above the grid. The possible values are "None" and "Standard". `objMyGrid.ViewsSelectorStyle = ViewsSelectorStyle.None;`
ShowSelectAllOption	Indicates if a check box is to be added in the grid for selecting all the records at a time. `objMyGrid.ShowSelectAllOption = true;`
ShowPreviewPane	Indicates if the Preview Pane has to be shown in the grid or not. `objMyGrid.ShowPreviewPane = true;`
FilterOperator	Operator to be used on the FilterField and FilterValue. `objMyGrid.FilterOperator = "eq";`
SaveUrl	To make the grid editable this property should be set to the url for which the modified data needs to be posted. `objMyGrid.Saveurl = "/c360sdk/Toolkit/SaveGridData.aspx?orgname=" c360UserContext.CurrentOrganizationName;`
AddInlineRow	Indicates if inline row addition shoulde be enabled in the grid or not. `objMyGrid.AddInlineRow = true;`

How to make the Grid Editable?

Inorder to display the grid in the editable mode 2 properties needs to be set. The 'SaveUrl' property should be set to the page for which the data needs to be posted. This should be pointing to '/c360sdk/Toolkit/SaveGridData.aspx?orgname=CurrentOrganizationName'. This page will take care of saving the grid changes. To enable the inline addition of the row in the grid, the property 'AddinlineRow' should be set to true.

Grid Server Side Methods

[There are no SDK-specific methods for this control]

Grid Client Side Properties

IMPORTANT NOTE: The naming convention for the grid's client side variables is "name of the property", followed by an underscore, and followed by the unique identifier of the Grid control that you specified in the ID property on the server side. For example, if you set the ID of you grid in your C# code to:

```
objMyGrid.ID = 'TestGrid';
```

There will be a JavaScript variable named:

```
iPageNumber_TestGrid
```

Containing the integer value of the data page currently displayed. The following table lists all the variables but omits the underscore and the ID as part of each name for the sake of brevity; do not forget to adjust your JavaScript code accordingly.

Grid (cont'd)

Grid Client Side Properties (cont'd)

Client Side Property (type)	**Description/Comments**
iPageNumber (int)	The number of the data page currently displayed.
iRecordCount (int)	The total number or records currently displayed on the page.
sViewId (string)	The unique identifier of the Saved Query.
sSortCol (string)	The name of the field that the data is currently sorted by.
sSortDir (string)	The direction that the data is currently sorted by.
sFetchXml (string)	The fetchXml used to fetch the data.
sFilterField (string)	The name of the field used to set filter criteria.
sFilterValue (string)	The value used to set filter criteria.
sJumpField (string)	The name of the field used to set filter criteria.
sJumpValue (string)	The value used to set filter criteria.
sAutoQueryApi (string)	The API method to call to retrieve the data.
sEntityId (string)	The unique identifier of the "parent" record.
iEntityTypeCode (int)	The entity type of the "parent" record.
iReturnedTypeCode (int)	The CRM type of the data being displayed.
bMoreRecords (bool)	Indicates if there are more data pages available.
bDistinctRecords (bool)	Indicates if duplicate records are being filtered.

Grid (cont'd)

Grid Client Side Methods

IMPORTANT NOTE: The naming convention for the grid's client side methods is "name of the method", followed by an underscore, and followed by the unique identifier of the Grid control that you specified in the ID property on the server side. For example, if you set the id of you grid in your C# code to:

```
objMyGrid.ID = 'TestGrid';
```

There will be a JavaScript method named:

```
RefreshGrid_TestGrid()
```

That allows you to refresh the data presented in the grid. The following table lists all the methods but omits the underscore and the ID as part of each name for the sake of brevity; do not forget to adjust your JavaScript code accordingly.

Client Side Method (return)	**Description/Comments**
GetRow(iRowNumber) (XmlNode)	Returns the specified row's data.
GetRowFromUniqueId(strUniquelIdentifier) (int)	Returns the row number for a given unique identifier. Returns null if no rows with the specified unique identifier can be found.
GetSelectedId(iRowNumber) (string)	Returns unique identifier of specified row.
GetSelectedType(iRowNumber) (int)	Returns the CRM type of specified row.
JumpBarClick(strJumpValue) (void)	Simulates the clicking on one of the items on the jump bar. The "strJumpValue" must contain one letter of the alphabet or the pound sign ("#").
NextPage() (void)	Advances to the next data page.
RefreshGrid() (void)	Refreshes the data page being presented.
PreviousPage() (void)	Moves to the previous data page.
Search() (void)	Simulates clicking on the "Search" button.
SelectAllRows() (void)	Selects all the rows in the grid.
UnSelectAllRows() (void)	Unselects all the rows in the grid

Grid (cont'd)

Grid Client Side Methods (cont'd)

Client Side Method (return)	**Description/Comments**

GetSelectedRows() (array)	Returns an array with one element for each row selected by the user. Returns null if no rows have been selected. Each element of this array contains a three element array where the first element holds the index of the row, the second element holds the unique identifier and the third element holds the CRM type of the row.
	
	Here is an example showing how this method can be used:
	`var aSelected = GetSelectedRows_gridSearch();");`
	`if (aSelected == null || aSelected.length == 0) {`
	`alert("No records have been selected");`
	`return;`
	`}`
	`for (int i = 0; i < aSelected.length; i++) {`
	`var sMsg = "Row number: " + i;`
	`sMsg += "\nThe index of the selected row is:" + aSelected[i][0];`
	`sMsg += "\nThe unique id of the selected row is:" + aSelected[i][1];`
	`sMsg += "\nThe type of the selected row is:" + aSelected[i][2];`
	`alert(sMsg);`
	`}`

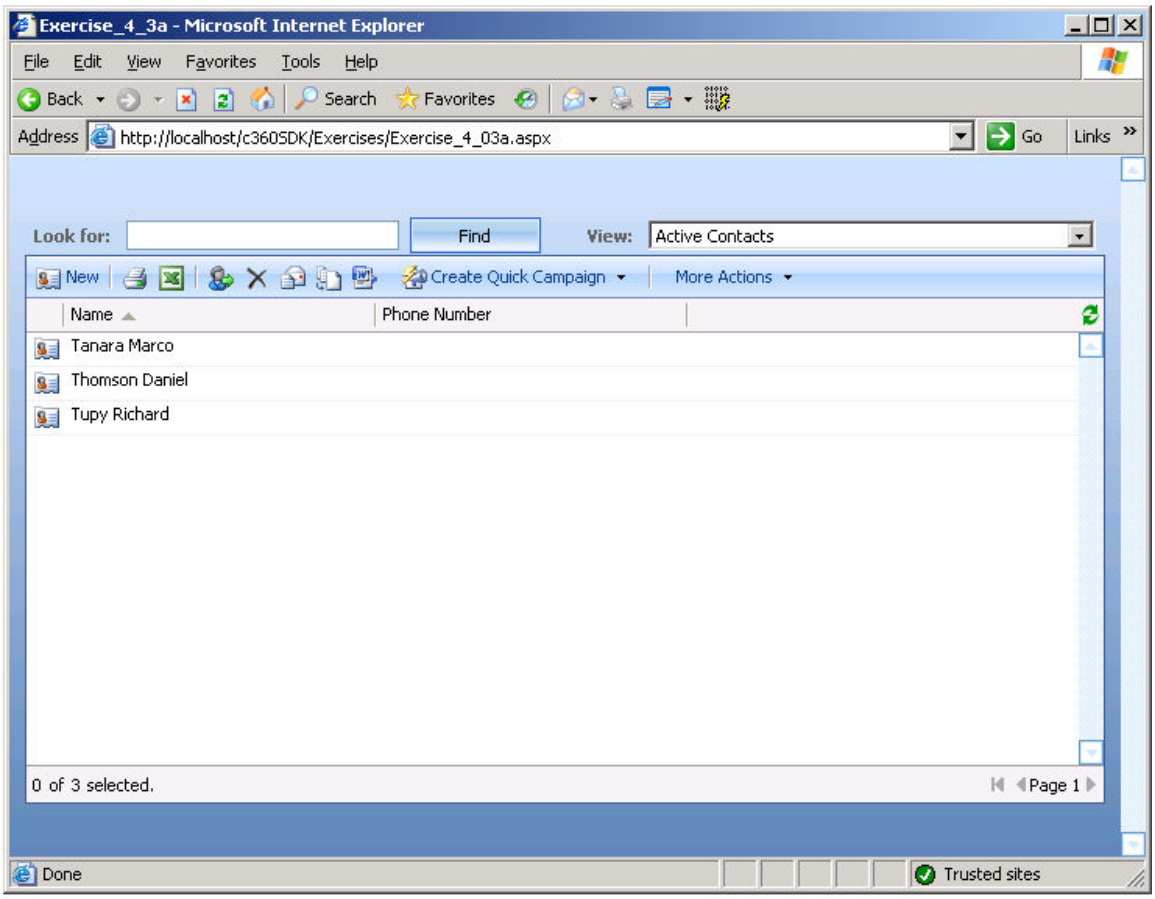
Grid (cont'd)

Exercise 4.3a: Grid

Build a page deriving from Area which:

- Displays Contacts in a grid
- Lists only Contacts with first names beginning with the letter "t"
- Displays the full name and phone number for each Contact
- Displays the advanced search tools
- Does not display the "Jump Bar"

Exercise 4.3a Desired Output



Grid (cont'd)

Exercise 4.3a Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildContentArea() {

                IGrid objGrid = ControlFactory.CreateGrid();
                objGrid.ID = "TestGrid";
                objGrid.ReturnedTypeCode = 2;
                objGrid.SortCol = "fullname";
                objGrid.SearchingStyle = SearchingStyles.Standard;
                objGrid.ViewsSelectorStyle = ViewsSelectorStyles.Standard;
                objGrid.ShowJumpBar = false;

                //fetch all users whose first name begins with the letter "t"
                objGrid.FetchXml = @"<fetch mapping='logical'>
                        <entity name='contact'>
                                <all-attributes/>
                                <filter type='and'>
                                        <condition attribute='fullname' operator='like' value='t%'/>
                                </filter>
                        </entity>
                </fetch>";

                //specify which columns you'd like to show for this search
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Name","fullname",200));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Phone Number","telephone1",200));

                Instance.ContentArea.Controls.Add(objGrid.Control);
        }
</script>
```

Exercise 4.3b: Grid & GridData

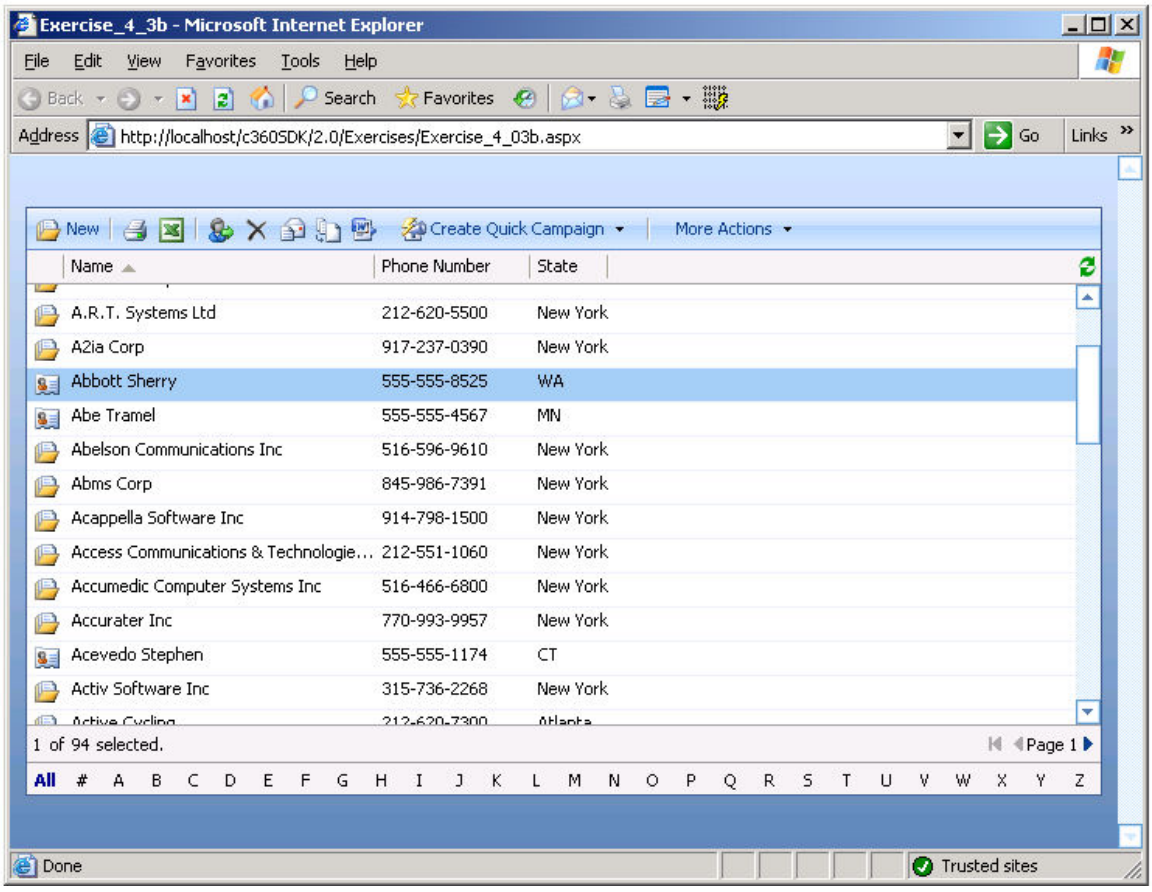
Build a page deriving from AreaPage which:

- Displays Contacts AND Accounts in a grid
- Displays the full name, phone number, and state for each Contact and the name, business phone number, and state for Accounts
- Does not display the search tool
- Displays the "Jump Bar"

Also, build a page deriving from GridData which:

- Filters the records such that only those located in the Unites States are displayed
- Filters the records such that only those located in states beginning with the letter clicked on by the user on the jump bar are displayed

Exercise 4.3b Desired Output



Grid (cont'd)

Exercise 4.3b Solution

Grid UI Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildContentArea() {
                IGrid objGrid = ControlFactory.CreateGrid();
                objGrid.ID = "TestGrid";
                objGrid.ReturnedTypeCode = 1;
                objGrid.SortCol = "fullname";
                objGrid.SearchingStyle = SearchingStyles.None;
                objGrid.ViewsSelectorStyle = ViewsSelectorStyles.None;
                objGrid.FetchDataPage = "Exercise_4_3b_GridData.aspx";

                //specify which columns you'd like to show for this search
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Name","fullname",200));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Phone Number","telephone1",100));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("State","address1_stateorprovince",50));

                Instance.ContentArea.Controls.Add(objGrid.Control);
        }

</script>
```

Exercise 4.3b Solution (cont'd)

GridData Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.GridDataPage" %>
<%@ Import Namespace="ToolkitSamples.MSCRM" %>

<script runat="server">
	public override void FetchData() {
		CrmService service = new CrmService();
		service.Credentials = System.Net.CredentialCache.DefaultCredentials;

		//first, let's convert the JumpMenu value to a usable one
		string filterValue = CleanFilterValue(Instance.JumpValue);

		// Retrieve all accounts in the USA
		string fetch = "<fetch mapping='logical'><entity name='account'><all-attributes/><filter type='and'>";
		fetch += "<condition attribute='address1_country' operator='eq' value='United States'/>";
		fetch += "<condition attribute='address1_stateorprovince' operator='like' value='";
		fetch += filterValue + "' />";
		fetch += "</filter></entity></fetch>";

		string strResultSet = service.Fetch(fetch);
		strResultSet = strResultSet.Replace("<name>", "<fullname>");
		strResultSet = strResultSet.Replace("<name/>", "<fullname/>");
		strResultSet = strResultSet.Replace("<name ", "<fullname ");
		strResultSet = strResultSet.Replace("</name>", "</fullname>");

		// Retrieve all contacts in the USA
		string fetch2 = "<fetch mapping='logical'><entity name='contact'><all-attributes/><filter type='and'>";
		fetch2 += "<condition attribute='address1_country' operator='eq' value='U.S.'/>";
		fetch2 += "<condition attribute='address1_stateorprovince' operator='like' value='";
		fetch2 += filterValue + "' />";
		fetch2 += "</filter></entity></fetch>";

		String strResultSet2 = service.Fetch(fetch2);
		Instance.SortCol = "fullname";

		AddResultSetToReturn(strResultSet, 1, "accountid");
		AddResultSetToReturn(strResultSet2, 2, "contactid");
	}
```

Grid (cont'd)

Exercise 4.3b Solution (cont'd)

```
        private string CleanFilterValue(string jumpMenuValue) {
                // Clean the filter value
                jumpMenuValue = jumpMenuValue.Trim();
                if (jumpMenuValue == "#") {
                        filterValue = "[0-9]%";
                } else {
                        string jumpMenuValueToLower = jumpMenuValue.ToLower();
                        string jumpMenuValueToUpper = jumpMenuValue.ToUpper();
                        if (jumpMenuValueToLower != fjumpMenuValueToUpper) {
                        filterValue = string.Concat("[", jumpMenuValueToLower, " | ", jumpMenuValueToUpper, "]%");
                        } else {
                                filterValue = jumpMenuValue + "%";
                        }
                }

                return filterValue;
        }
</script>
```

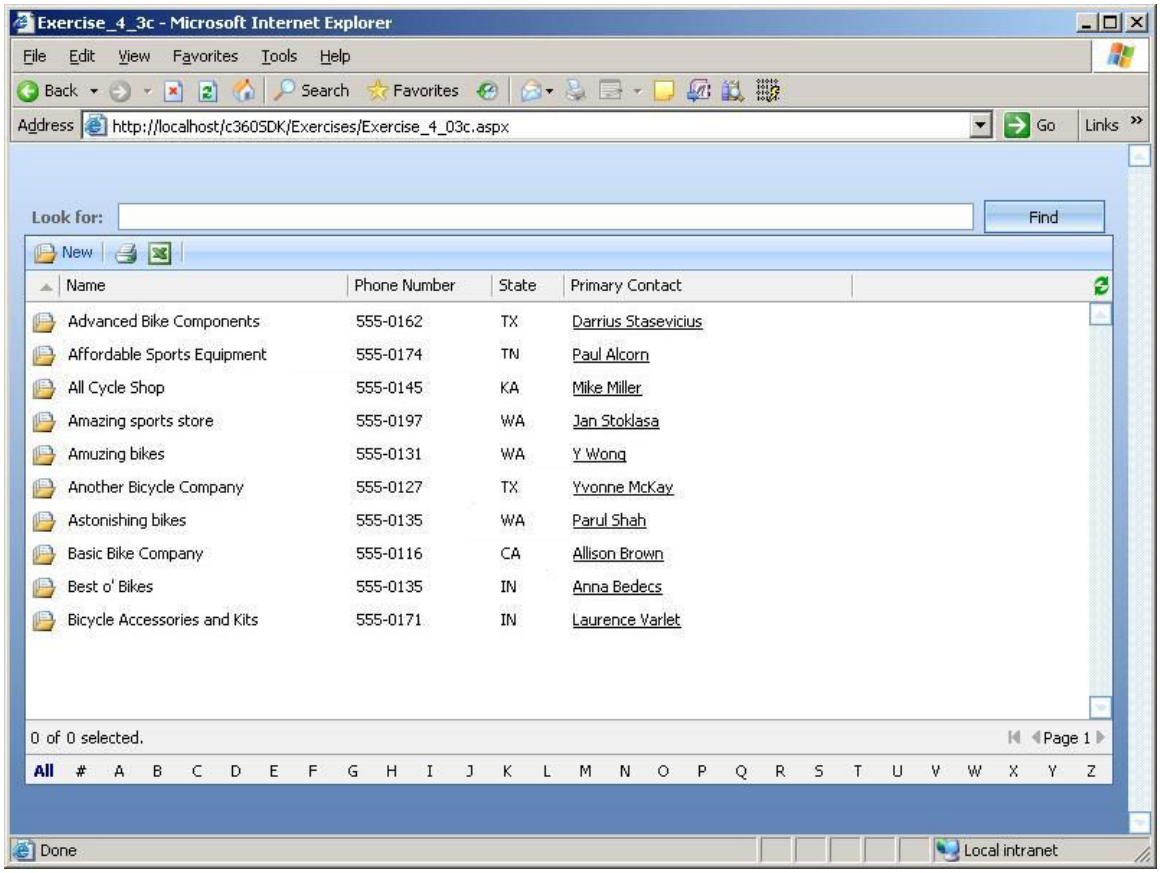
Exercise 4.3c: Grid Control with linked Entities

Build a page deriving from AreaPage which:

- Displays all U.S. Accounts in a grid
- Displays the account name, phone number, and primary contact
- Has the primary contact linked to the record
- Does not display the standard actions
- Does not display the search tool
- Displays the "Jump Bar"

Note: When using linked entities, first, the entity must be a lookup type in CRM. Also, the Grid control must have the appropriate property set, and the column which contains the linked entity must be defined as so (by setting the ColumnType)

Exercise 4.3c Desired Output



Grid (cont'd)

Exercise 4.3c Solution

Grid UI Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildContentArea() {
                IGrid objGrid = ControlFactory.CreateGrid();
                objGrid.ID = "TestGrid";
                objGrid.ReturnedTypeCode = 1;
                objGrid.SortCol = "fullname";
                objGrid.SearchingStyle = SearchingStyles.None;
                objGrid.ViewsSelectorStyle = ViewsSelectorStyles.None;
                objGrid.FetchDataPage = "Exercise_4_3c_GridData.aspx";
                objGrid.EnableEntityLink = true;
                objGrid.IncludeStandardActions = false;

                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Name","name",200));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Phone Number","telephone1",100));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("State","address1_stateorprovince",50));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Primary Contact", "primarycontactid", 200,
ColumnType.EntityLink));

                Instance.ContentArea.Controls.Add(objGrid.Control);
        }

</script>
```

Exercise 4.3c Solution (cont'd)

GridData Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.GridDataPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.CRM" %>

<script runat="server">
	public override void FetchData() {
		//first, let's convert the JumpMenu value to a usable one
		string filterValue = CleanFilterValue(Instance.JumpValue);

		// Retrieve all accounts in the USA
		string fetch = "<fetch mapping='logical'><entity name='account'><all-attributes/><filter type='and'>";
		fetch += "<condition attribute='address1_country' operator='eq' value='U.S.'/>";
		fetch += "<condition attribute='address1_stateorprovince' operator='like' value='";
		fetch += filterValue + "' />";
		fetch += "</filter></entity></fetch>";
		string strResultSet = CrmService.ExecuteFetchXml(fetch);

		// Set the default sort column
		Instance.SortCol = "name";

		// Return the result to be rendered
		AddResultSetToReturn(strResultSet, 1, "accountid");
	}

	private string CleanFilterValue(string jumpMenuValue) {
		// Clean the filter value
		string filterValue;
		jumpMenuValue = jumpMenuValue.Trim();
		if (jumpMenuValue == "#") {
			filterValue = "[0-9]%";
		} else {
			string jumpMenuValueToLower = jumpMenuValue.ToLower();
			string jumpMenuValueToUpper = jumpMenuValue.ToUpper();
			if (jumpMenuValueToLower != jumpMenuValueToUpper) {
				filterValue = string.Concat("[", jumpMenuValueToLower, " | ", jumpMenuValueToUpper, "]%");
			} else {
				filterValue = jumpMenuValue + "%";
			}
		}

		return filterValue;
	}
</script>
```

Grid (cont'd)

Exercise 4.3d: Using the Grid Control and DataTable data source

Build a page deriving from AreaPage which:

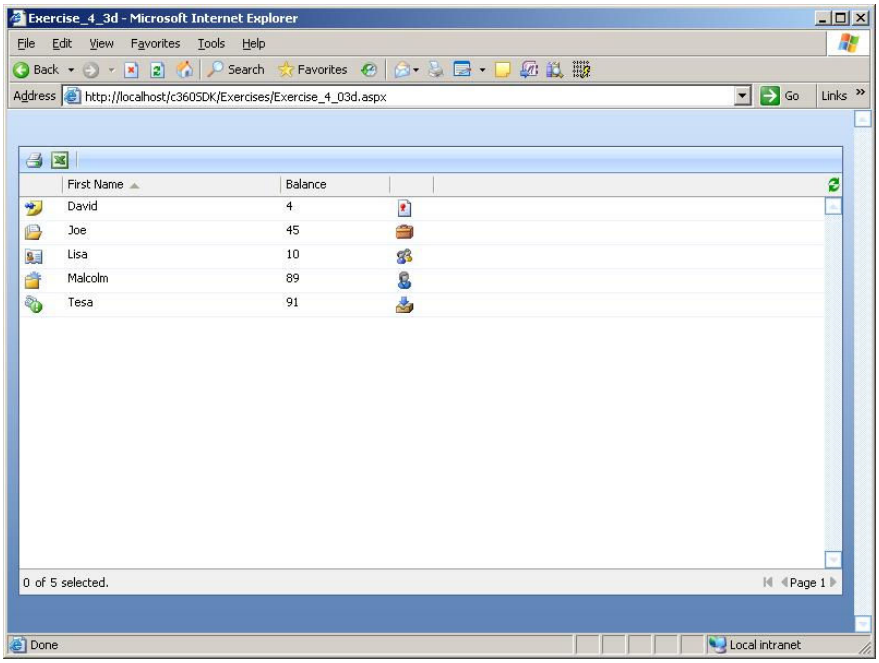
- Contains a 4 column grid
  - an icon of the record (using the "Image" ColumnType)
  - first name
  - balance
  - another icon (also using the "Image" ColumnType)
- Does not display the standard actions
- Does not display the "Create New" button
- Does not display the search tool
- Does not display the "Jump Bar"

Also, build a page deriving from GridData which:

- Contains a sample DataTable with five columns for the primary key (int), icon (string), firstname (string), balance (int) and another icon (string).
- Contains 5 sample rows of data

Note: When using a DataTable as your data source of the grid, it is important to remember that the field names are case-sensitive. Also, when creating a Grid column of type Image, the value of the field must be the location of the image.

Exercise 4.3d Desired Output



Grid (cont'd)

Exercise 4.3d Solution

Grid UI Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildContentArea() {
                IGrid objGrid = ControlFactory.CreateGrid();
                objGrid.ID = "TestGrid";
                objGrid.ReturnedTypeCode = 1;
                objGrid.ShowEntityIcon = false;
                objGrid.SortCol = "name";
                objGrid.SearchingStyle = SearchingStyles.None;
                objGrid.ViewsSelectorStyle = ViewsSelectorStyles.None;
                objGrid.FetchDataPage = "Exercise_4_03d_GridData.aspx";

                objGrid.IncludeStandardActions = false;
                objGrid.AllowCreateNew = false;
                objGrid.ShowJumpBar = false;

                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("", "icon", 40, ColumnType.Image));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("First Name", "name", 200));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Balance", "balance", 100));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("", "icon2", 40, ColumnType.Image));

                Instance.ContentArea.Controls.Add(objGrid.Control);
        }
</script>
```

Grid (cont'd)

Exercice 4.3d Solution (cont'd)

GridData Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.GridDataPage" Language="C#" debug="true" %>
<%@ Import Namespace="c360.Toolkit.SDK.WebServices" %>
<%@ Import Namespace="System.Data" %>

<script runat="server">
        public override void FetchData() {
                System.Data.DataTable dt = new System.Data.DataTable();

                dt.Columns.Add("primarykey", typeof(int));
                dt.Columns.Add("icon", typeof(string));
                dt.Columns.Add("name", typeof(string));
                dt.Columns.Add("balance", typeof(int));
                dt.Columns.Add("icon2", typeof(string));

                dt.Rows.Add(new object[] { 1, "~/Toolkit/Images/ico_16_1.gif", "Joe", 45, "~/Toolkit/Images/ico_16_10.gif" });
                dt.Rows.Add(new object[] { 2, "~/Toolkit/Images/ico_16_2.gif", "Lisa", 10, "~/Toolkit/Images/ico_16_9.gif" });
                dt.Rows.Add(new object[] { 3, "~/Toolkit/Images/ico_16_3.gif", "Malcolm", 89, "~/Toolkit/Images/ico_16_8.gif" });
                dt.Rows.Add(new object[] { 4, "~/Toolkit/Images/ico_16_4.gif", "Tesa", 91, "~/Toolkit/Images/ico_16_2020.gif" });
                dt.Rows.Add(new object[] { 5, "~/Toolkit/Images/ico_16_5.gif", "David", 4, "~/Toolkit/Images/ico_16_1010.gif" });

                Instance.SortCol = "name";

                Instance.AddDataTableToReturn(dt, 0, "primarykey");
        }
</script>
```

Exercise 4.3e: Grid Control with external links

Build a page deriving from AreaPage which:

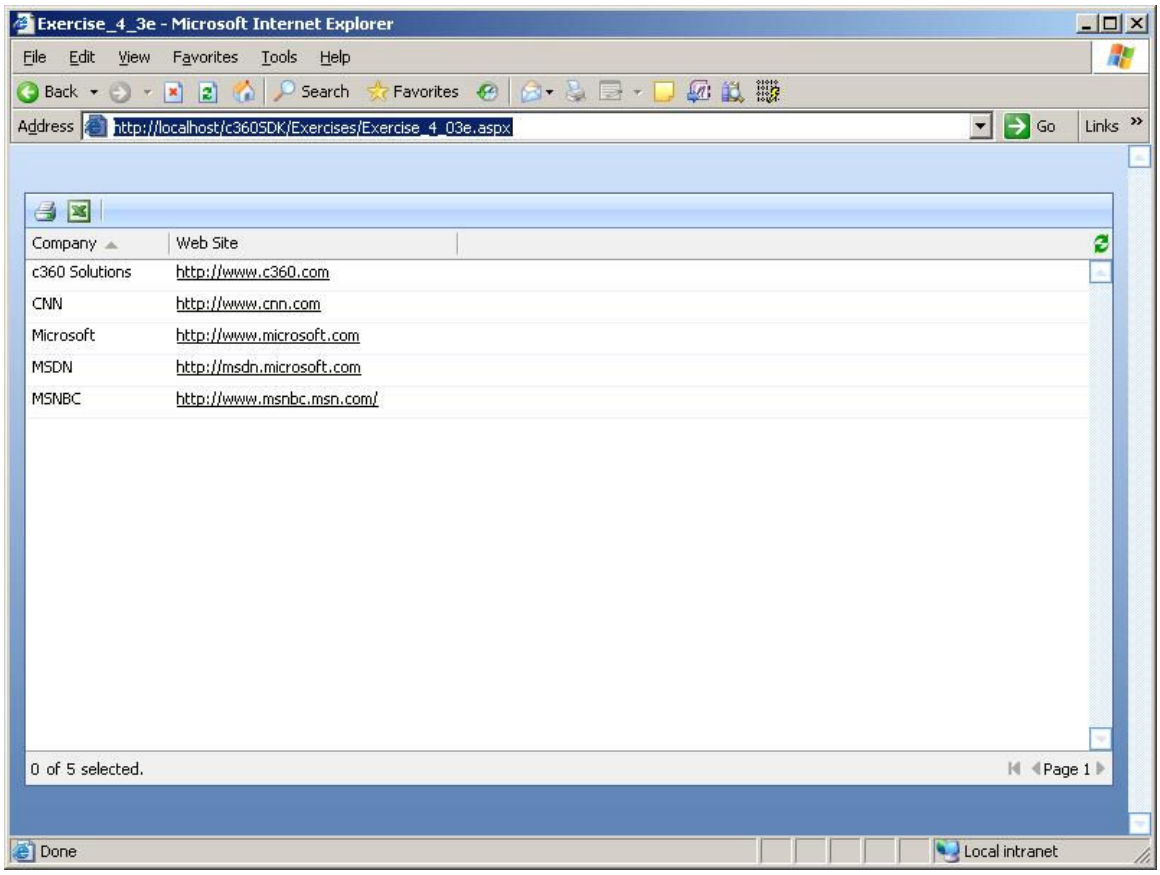
- Displays a list of 5 companies
- Displays the name of the company and a link to their web site
- Does not display the standard actions
- Does not display the "Create New" button
- Does not display the search tool
- Does not display the "Jump Bar"

Also, build a page deriving from GridData which:

- Contains a sample DataTable with three columns for the primary key (int), company name (string) and web site.
- Contains 5 sample rows of data

Note: When using a DataTable as your data source of the grid, it is important to remember that the field names are case-sensitive.

Exercise 4.3e Desired Output



Grid (cont'd)

Exercise 4.3e Solution

Grid UI Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildContentArea() {
                IGrid objGrid = ControlFactory.CreateGrid();
                objGrid.ID = "TestGrid";
                objGrid.ReturnedTypeCode = 0;
                objGrid.ShowEntityIcon = false;
                objGrid.SortCol = "companyname";
                objGrid.SearchingStyle = SearchingStyles.None;
                objGrid.ViewsSelectorStyle = ViewsSelectorStyles.None;
                objGrid.FetchDataPage = "Exercise_4_03e_GridData.aspx";

                objGrid.IncludeStandardActions = false;
                objGrid.AllowCreateNew = false;
                objGrid.ShowJumpBar = false;

                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Company", "companyname", 0, ColumnType.Normal));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Web Site", "website", 200, ColumnType.ExternalLink));

                Instance.ContentArea.Controls.Add(objGrid.Control);
        }
</script>
```

Exercice 4.3e Solution (cont'd)

GridData Page:

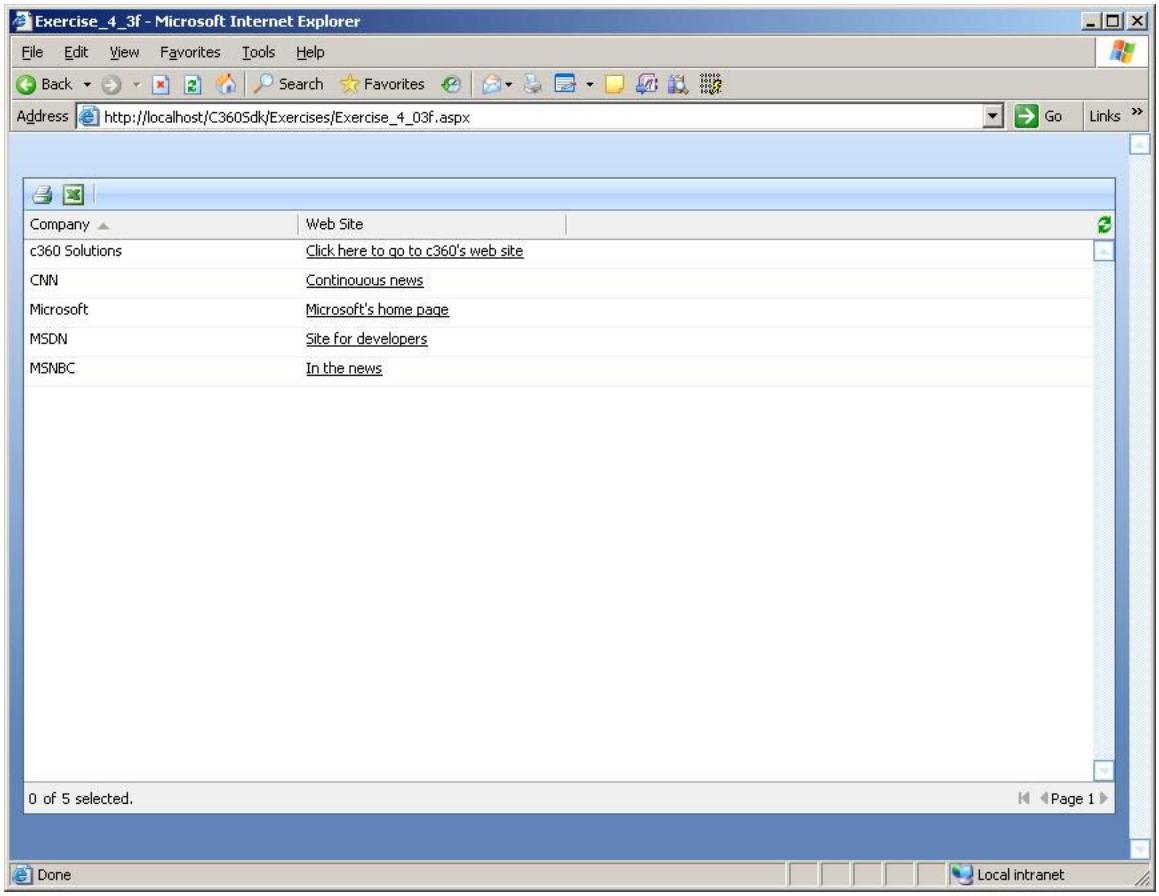
```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.GridDataPage" Language="C#" debug="true" %>
<%@ Import Namespace="c360.Toolkit.SDK.WebServices" %>
<%@ Import Namespace="System.Data" %>

<script runat="server">
        public override void FetchData() {
                System.Data.DataTable dt = new System.Data.DataTable();

                dt.Columns.Add("primarykey", typeof(int));
                dt.Columns.Add("companyname", typeof(string));
                dt.Columns.Add("website", typeof(string));

                dt.Rows.Add(new object[] { 1, "c360 Solutions", "http://www.c360.com" });
                dt.Rows.Add(new object[] { 2, "Microsoft", "http://www.microsoft.com" });
                dt.Rows.Add(new object[] { 3, "MSDN", "http://msdn.microsoft.com" });
                dt.Rows.Add(new object[] { 4, "CNN", "http://www.cnn.com" });
                dt.Rows.Add(new object[] { 5, "MSNBC", "http://www.msnbc.msn.com/" });

                Instance.AddDataTableToReturn(dt, 0, "primarykey");
        }
</script>
```

Exercise 4.3f: Advanced Grid Control with external links

Improve the previous exercise by:

- Specifying a caption for each link
- Specifying a height and width for each link
- Making sure each link opens in its own window instead of reusing one window

Exercise 4.3f Desired Output



Grid (cont'd)

Exercise 4.3f Solution

Grid UI Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildContentArea() {
                IGrid objGrid = ControlFactory.CreateGrid();
                objGrid.ID = "TestGrid";
                objGrid.ReturnedTypeCode = 0;
                objGrid.ShowEntityIcon = false;
                objGrid.SortCol = "companyname";
                objGrid.SearchingStyle = SearchingStyles.None;
                objGrid.ViewsSelectorStyle = ViewsSelectorStyles.None;
                objGrid.FetchDataPage = "Exercise_4_03f_GridData.aspx";

                objGrid.IncludeStandardActions = false;
                objGrid.AllowCreateNew = false;
                objGrid.ShowJumpBar = false;

                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Company", "companyname", 0, ColumnType.Normal));
                objGrid.ColumnHeaders.Add(ControlFactory.CreateColumnHeader("Web Site", "website", 200, ColumnType.ExternalLink));

                Instance.ContentArea.Controls.Add(objGrid.Control);
        }
</script>
```

Exercice 4.3f Solution (cont'd)

GridData Page:

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.GridDataPage" Language="C#" debug="true" %>
<%@ Import Namespace="c360.Toolkit.SDK.WebServices" %>
<%@ Import Namespace="System.Data" %>

<script runat="server">
        public override void FetchData() {

                StringBuilder resultset = new StringBuilder();
                resultset.Append("<resultset>");
                resultset.Append("<result>");
                resultset.Append("<primarykey>1</primarykey>");
                resultset.Append("<companyname>c360 Solutions</companyname>");
                resultset.Append("<website url='http://www.c360.com' width='800' height='600' windowName='c360'>Click here to
go to c360's web site</website>");
                resultset.Append("</result>");
                resultset.Append("<result>");
                resultset.Append("<primarykey>2</primarykey>");
                resultset.Append("<companyname>Microsoft</companyname>");
                resultset.Append("<website url='http://www.Microsoft.com' width='700' height='500'
windowName='MSFT'>Microsoft's home page</website>");
                resultset.Append("</result>");
                resultset.Append("<result>");
                resultset.Append("<primarykey>3</primarykey>");
                resultset.Append("<companyname>MSDN</companyname>");
                resultset.Append("<website url='http://msdn.microsoft.com' width='600' height='400' windowName='MSDN'>Site for
developers</website>");
                resultset.Append("</result>");
                resultset.Append("<result>");
                resultset.Append("<primarykey>4</primarykey>");
                resultset.Append("<companyname>CNN</companyname>");
                resultset.Append("<website url='http://www.cnn.com' width='500' height='300' windowName='CNN'>Continuous
news</website>");
                resultset.Append("</result>");
                resultset.Append("<result>");
                resultset.Append("<primarykey>5</primarykey>");
                resultset.Append("<companyname>MSNBC</companyname>");
                resultset.Append("<website url='http://www.msnbc.msn.com' width='400' height='200' windowName='MSNBC'>In
the news</website>");
                resultset.Append("</result>");
                resultset.Append("</resultset>");

                Instance.AddResultSetToReturn(resultset.ToString(), 0, "primarykey");
        }
</script>
```

GridPreferences

Overview

The GridPreferences control allows you to prompt users to select a list of fields that they want to see inside of a grid, as well as selecting a default sort field and sort direction. This control displays a dual-list with all of the available fields on the left, and the selected fields on the right, as well as a drop down list for the user to define the default sort field and direction.

In order to use a GridPreferences control to control a grid, you would need to have the GridPreferences page linked to from the GridMenu. Then, you will have to create a function to save the data to an XML file or a database when the user chooses their options and hits "OK" on the popup. This can be accomplished by overriding the SaveData method of the Save Dialog page (from which this page should be inherited).

GridPreferences Server Side Properties

Server Side Property (type)	**Description/Comments**
AllFields (AttributeMetadata[])	A list of all of the available fields to be presented in the left list.
Height (Unit)	The height of the control. `objMyGridPreferences.Height = Unit.Percentage(100);`
ID (String)	The unique ID of the control. `objMyGridPreferences.ID = "myPreferences";`
SelectedFields (AttributeMetadata[])	A list of all fields currently selected by the user.
SelectedSortByField (string)	The user defined sort field. `objMyGridPreferences.SortByField = "createdon";`
SelectedSortDirection (OrderType)	The user defined sort direction. The possible values are "Ascending" and "Descending". `objMyGridPreferences.SelectedSortDirection = OrderType.Descending;`
Width (Unit)	The width of the control. `objMyGridPreferences.Width = Unit.Percentage(100);`

GridPreferences (cont'd)

GridPreferences Server Side Methods

[There are no SDK-specific methods for this control]

GridPreferences Client Side Properties

NOTE: on the client side, there is no GridPreference control per-se but rather 4 distinct controls:

- A list containing all the available fields called "<unique ID>_Fields_LeftList".
- A list containing the selected fields called "<unique ID>_Fields_RightList".
- A drop-down select control allowing the user to specify the sort field called "<Unique ID>_SortBySelect"
- A drop-down select control allowing the user to specify the sort direction called "<Unique ID>_SortDirectionSelect"

Here is an example showing how you can count the number of items in a list.

```
Alert("The number of selected fields is: " + myGridPreferences_Fields_LeftList.GetAllValues().length);
```

For more details on the client-side properties of the two lists, please see the "List" control.

GridPreferences Client Side Methods

IMPORTANT NOTE: The naming convention for the GridPreferences' client-side methods is "name of the method", followed by an underscore, and followed by the unique identifier of the GridPreferences control that you specified in the ID property on the server side. See the "Grid" control for more details and samples.

Client Side Method (return)	**Description/Comments**
fillSortFieldsSelect() (array)	Updates the "SortBy" drop-down select to offer the appropriate list of fields. This method is automatically invoked when the user adds or removes a field from the list of selected fields but you can invoke it if you programmatically add and/or remove items from the same list without the user's intervention.

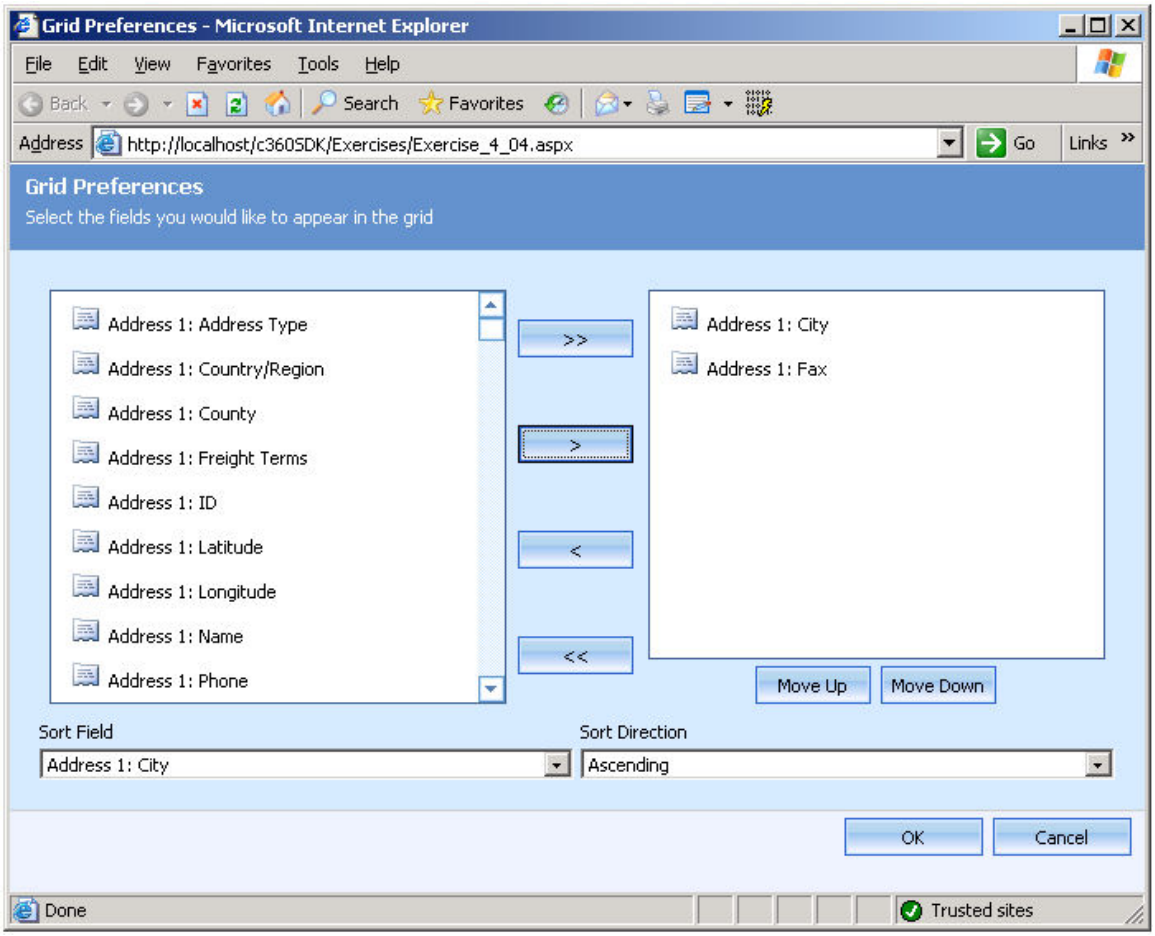
GridPreferences (cont'd)

Exercise 4.4: GridPreferences

Build a page deriving from save which:

- Displays a GridPreferences control which the user can control which fields to display in the grid for a contact on the left.
  - NOTE: To dynamically generate a list of all the available fields for an entity, you can query the MSCRM metadata.
- The save dialog contains the "Ok" and "Cancel" buttons and has a title of "Grid Preferences"

Exercise 4.4 Desired Output



GridPreferences (cont'd)

Exercise 4.4 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.Dialogs.SavePage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>
<%@ Import Namespace="c360.Toolkit.CRM" %>

<script runat="server">
	public override void BuildMenuArea(){
		Instance.ClientSidePreSaveHandler = "SaveHandler";
		Instance.DialogTitle = "Grid Preferences ";
		Instance.DialogDescription = "Select the fields you would like to appear in the grid";
		base.BuildMenuArea();
	}

	public override void BuildContentArea() {
		IGridPreferences objGridPreferences = ControlFactory.CreateGridPreferences();

		AttributeMetadata[] attributes = Instance.MetadataService. RetrieveEntityMetadata("contact").Attributes;

		//the attributes array contains values that we don't want (such as blank display names), so to filter it out
		//1.we must determine the count of attributes with DisplayNames
		int iValidAttributeCount = 0;
		for(int j = 0; j<attributes.Length; j++) {
			if (attributes[j].DisplayName != "" && attributes[j].DisplayName != null) {
				iValidAttributeCount++;
			}
		}

		//2.create a new AttributeMetadata array with a length of the  iValidAttributeCount
		AttributeMetadata[] validAttributes = new AttributeMetadata[iValidAttributeCount];
		int iNew = 0;

		//3. Fill the new AttributeMetadata array with the valid attributes
		for (int i = 0; i < attributes.Length; i++) {
			if (attributes[i].DisplayName != "" && attributes[i].DisplayName != null) {
				validAttributes[iNew] = attributes[i];
				iNew++;
			}
		}
		objGridPreferences.AllFields = validAttributes;

		Instance.ContentArea.Controls.Add(objGridPreferences.Control);
	}

	public override void BuildFooterArea(){
		Instance.AddOkButton("btnOk");
		Instance.AddCancelButton("btnCancel");
	}
</script>
```

LeftNavBar

Overview

This control mimics the navigation area in the left part of the Microsoft CRM edit window. The LeftNavBar is composed of NavBarItems, each with their own icon, tooltip and action.

LeftNavBar Server Side Properties

Server Side Property (type)	**Description/Comments**
ID (String)	The unique ID of the control. `objMyLeftNav.ID = "myLeftNav";`

Items (NavBarItemCollection)	Allows you to add items to the left navigation bar. When adding a tab, you must specify the following 3 properties:
	· The caption of the tab
	· The icon location
	· The tooltip to display when mouse-over
	
	You can add Items on the LeftNavBar by adding to the Items collection like in the following example:
	`ITabBar objMyLeftNav = ControlFactory.CreateLeftNavBar();`
	`objMyLeftNav.Items.Add(ControlFactory.CreateTabItem("General", "/images/ico.gif","The General Area"));`
	`objMyLeftNav.Items.Add(ControlFactory.CreateTabItem("Advanced", "/images/ico2.gif","The Advanced Area"));`

LeftNavBar Server Side Methods

Server Side Methods (return)	**Description/Comments**

LoadFromXmlFile(string xmlFile**)**	The user can specify an xml file which looks like the sample xml listed below.
	
	`<root>`
	`<child text="Math" iconPath="" action="" tooltip="Math"></child>`
	`<child text="History" iconPath="" action="" tooltip="History"></child>`
	`<child text="NewItem" iconPath="month_icon.JPG" action="displayNavArea(4,'http://localhost/C360SDK/Exercises/Exercise_4_03a.aspx')" tooltip="NewItem"></child>`
	`</root>`
	· text – Name of the node on LeftNavBar
	· action – Navigate you corresponding page
	· iconPath – The icon location
	· tooltip - The tooltip to display when mouse-over
	
	The name of the root node or the child node can be anything, but the attributes for the child node has to be the one specified in the above sample.

LeftNavBar (cont'd)

LeftNavBar Client Side Properties

[There are no SDK-specific properties for this control]

LeftNavBar Client Side Methods

Client Side Method (return)	**Description/Comments**

displayNavArea() (void)	This JavaScript function is best used in the "Action" of NavBar items to update the content area when a user clicks on the NavItem in question. Here is a sample server-side code where this function is specified as the action (or click handler if you prefer) for three items:
	
	public override void BuildLeftArea() {
	ILeftNavBar navBar = ControlFactory.CreateLeftNavBar();
	navBar.Items.Add(ControlFactory.CreateNavBarItem("First", "/imgs/1.gif", "displayNavArea(0, '1.aspx')", "Tooltip 1"));
	navBar.Items.Add(ControlFactory.CreateNavBarItem("Second", "/imgs/2.gif", "displayNavArea(1, '2.aspx')", "Tooltip 1"));
	navBar.Items.Add(ControlFactory.CreateNavBarItem("Third", "/imgs/3.gif", "displayNavArea(2, '3.aspx')", "Tooltip 1"));
	}

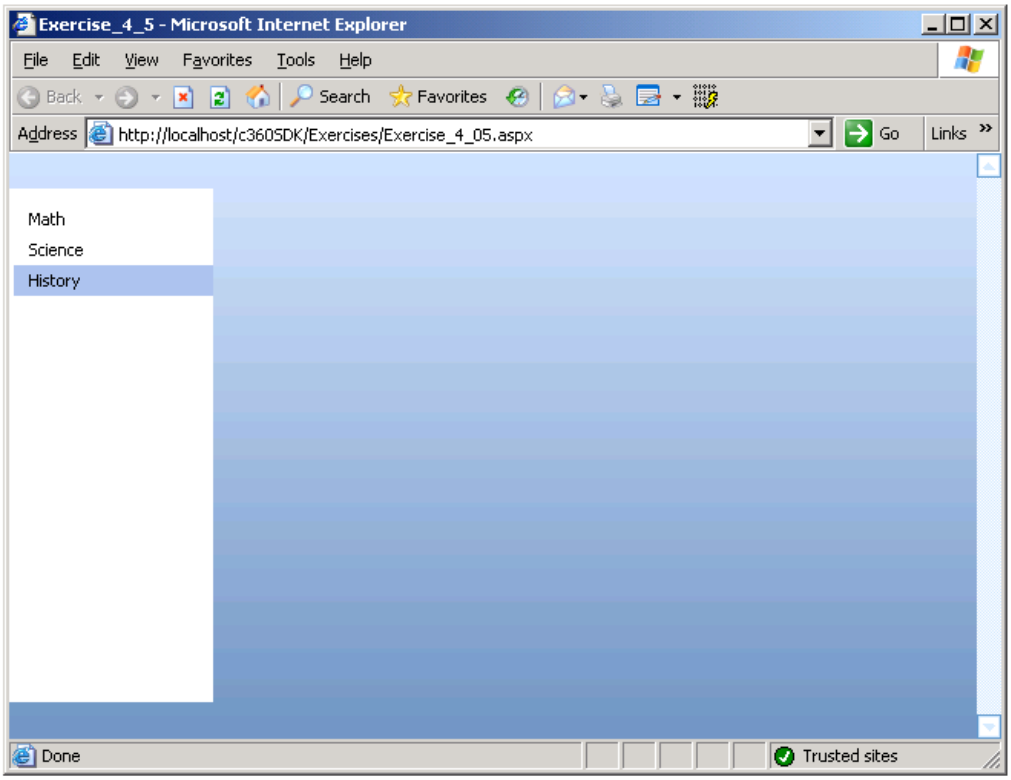
LeftNavBar (cont'd)

Exercise 4.5: LeftNavBar

Create a page deriving from Area which:

- The LeftArea has a white background color
- Contains a LeftNavBar in the LeftArea of the page
- The LeftNavBar to contain three subjects taught in school
- The second subject on the list is selected by default

Exercise 4.5 Desired Output



Exercise 4.5 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildLeftArea() {
                ILeftNavBar objLeftNavBar = ControlFactory.CreateLeftNavBar();
                objLeftNavBar.Items.Add(ControlFactory.CreateNavBarItem("Math", "", "", "Math", false));
                objLeftNavBar.Items.Add(ControlFactory.CreateNavBarItem("Science", "", "", "Science", true));
                objLeftNavBar.Items.Add(ControlFactory.CreateNavBarItem("History", "", "", "History", false));
                Instance.LeftArea.BackColor = System.Drawing.Color.White;
                Instance.LeftArea.Controls.Add(objLeftNavBar.Control);
        }
</script>
```

List

Overview

The list control is used to display a list of items. When listing items that represent CRM data, the list can automatically add the appropriate icon next to each item. Optionally, the user can be allowed to sort the items by clicking on the “Up” or “Down” button.

List Server Side Properties

Server Side Property (type)	**Description/Comments**
AllowMoveItemsDown (bool)	Indicates if the user is authorized to re-sort items by clicking on the “Move Down” button. `myList.AllowMoveItemsDown = true;`
AllowMoveItemsUp (bool)	Indicates if the user is authorized to re-sort items by clicking on the “Move Up” button. `myList.AllowMoveItemsUp = true;`
AllowMultiSelect (bool)	Indicates if the user can select more than one item. `myList.AllowMultiSelect = false;`

ClickEventHandler (string)	Name of a JavaScript function that will be called when the user clicks on an item. The JavaScript function you specify will be passed an "EventObject" parameter. This EventObject has the following properties:
	• itemIndex: a numeric value that represents the position of the item clicked
	• itemValue: unique identifier of the item clicked
	• itemType: CRM object type of the item clicked
	• itemCaption: the caption of the item clicked
	• itemLevel: the indentation level of the item that was clicked
	`function ClickHandler() {`
	`  var strItemIndex = event.itemIndex;`
	`  var strItemValue = event.itemValue;`
	`  var strItemType = event.itemType;`
	`  var strItemCaption = event.itemCaption;`
	`  var strItemLevel = event.itemLevel;`
	`  alert("You just clicked on " + strItemCaption);`
	`}`

DoubleClickEventHandler (string)	Name of a JavaScript function that will be called when the user double-clicks on an item. The JavaScript function you specify will be passed an "EventObject" parameter. This EventObject has the same properties as for the click event. Refer to the ClickEventHandler property for more details and for a code sample
Height (Unit)	The height of the rendered List. `myList.Height = Unit.Percentage(100);`

List (cont'd)

Server Side Property (type)	Description/Comments
ID (String)	The unique ID of the control. `objMyList.ID = "myList";`

Items (ListItemsCollection)	Collection of list items. When adding a list item, you must specify the following properties:
	· The caption of the item
	· The CRM object type of the item
	· The value of this item
	· A boolean indicating whether this item can be clicked on (and therefore can be selected) by the user or not
	· The path to the icon to be displayed next to this item
	· The indentation level of this item
	· A boolean indicating if this item is selected by default or not
	Some of these properties are optional and can be omitted. For example, if you don't specify a value for the iconPath, the CRM icon corresponding to the CRM object type you specified will be automatically displayed.
	
	You can add list items on the list control by adding to the Items collection like in the following example:
	`List lstTest = new List();`
	`lstTest .ID = "Testing";`
	`lstTest .Items.Add(ControlFactory.CreateListItem("Testing", 1, "xyz"));`

Sorted (bool)	Indicates if the items in the list will be automatically sorted alphabetically by comparing the Caption of the ListItems with each other. `myList.Sorted = true;`
Width (Unit)	The width of the rendered List. `myList.Width = Unit.Percentage(100);`

List Server Side Methods

[There are no SDK-specific methods for this control]

List (cont'd)

List Client Side Properties

Assuming that you have set the ID property to a unique value (let's say "MyList" for example), you can get access to the list object in your JavaScript code like this:

```
var objMyList = document.all.MyList;
```

After you have reference to this object, you have access to the following properties:

Client Side Property (type)	**Description/Comments**
allowMultiSelect (bool)	Client side equivalent of the AllowMultiSelect server-side property.
sorted (bool)	Client side equivalent of the Sorted server-side property.

List Client Side Methods

Client Side Method (return)	**Description/Comments**
AddItem (void)	Allows you to dynamically add an item at the bottom of the list. When adding an item, you must specify the value of this item, the CRM type and the caption of the item.
AddItemAt (void)	Allows you to dynamically add an item at a specified position in the list. This can be used, for example, to add an item at the top or the middle of the list. When adding an item, you must specify the value of this item, the CRM type, the caption and the position in the list where you want the item to be displayed. Optionally, you can also specify an indentation level.
ClearSelection (void)	Allows you to unselect all items from the list.
RemoveAll (void)	Allows you to remove all items from the list.
RemoveByIndex (void)	Allows you to remove an item from the list by specifying its position in the list.
RemoveByValue (void)	Allows you to remove an item from the list by specifying its value. If there is more than one item with the same value, only the first one will be removed.
GetAllCaptions (array)	Returns an array where each element contains the caption of an item present in the list. Returns null if the list does not contain any items.
GetAllIndexes (array)	Returns an array where each element contains the index of an item present in the list. Returns null if the list does not contain any items.
GetAllLevels (array)	Returns an array where each element contains the indentation level of an item present in the list. Returns null if the list does not contain any items.

List (cont'd)

List Client Side Methods (cont'd)

Client Side Method (return)	Description/Comments
GetAllTypes (array)	Returns an array where each element contains the CRM type of an item present in the list. Returns null if the list does not contain any items.
GetAllValues (array)	Returns an array where each element contains the value of an item present in the list. Returns null if the list does not contain any items.
GetSelectedCaptions (array)	Returns an array where each element contains the caption of an item selected by the user. It returns null if no items have been selected.
GetSelectedIndexes (array)	Returns an array where each element contains the index of an item selected by the user. Returns null if no items have been selected.
GetSelectedLevels (array)	Returns an array where each element contains the indentation level of an item selected by the user. It returns null if no items have been selected.
GetSelectedTypes (array)	Returns an array where each element contains the CRM type of an item selected by the user. It returns null if no items have been selected.
GetSelectedValues (array)	Returns an array where each element contains the value of an item selected by the user. Returns null if no items have been selected.
MoveSelectedItemDown (void)	Moves the selected item down in the list. The item will not be moved if it's already at the bottom of the list. Throws error if more than one item is selected.
MoveSelectedItemUp (void)	Moves the selected item up in the list. The item will not be moved if it's already at the top of the list. Throws an error if more than one item is selected.
SelectAll (void)	Selects all of the items in the list.
SelectByIndex (void)	Selects an item on the list by index position.
SelectByValue (void)	Selects an item on the list by its value.

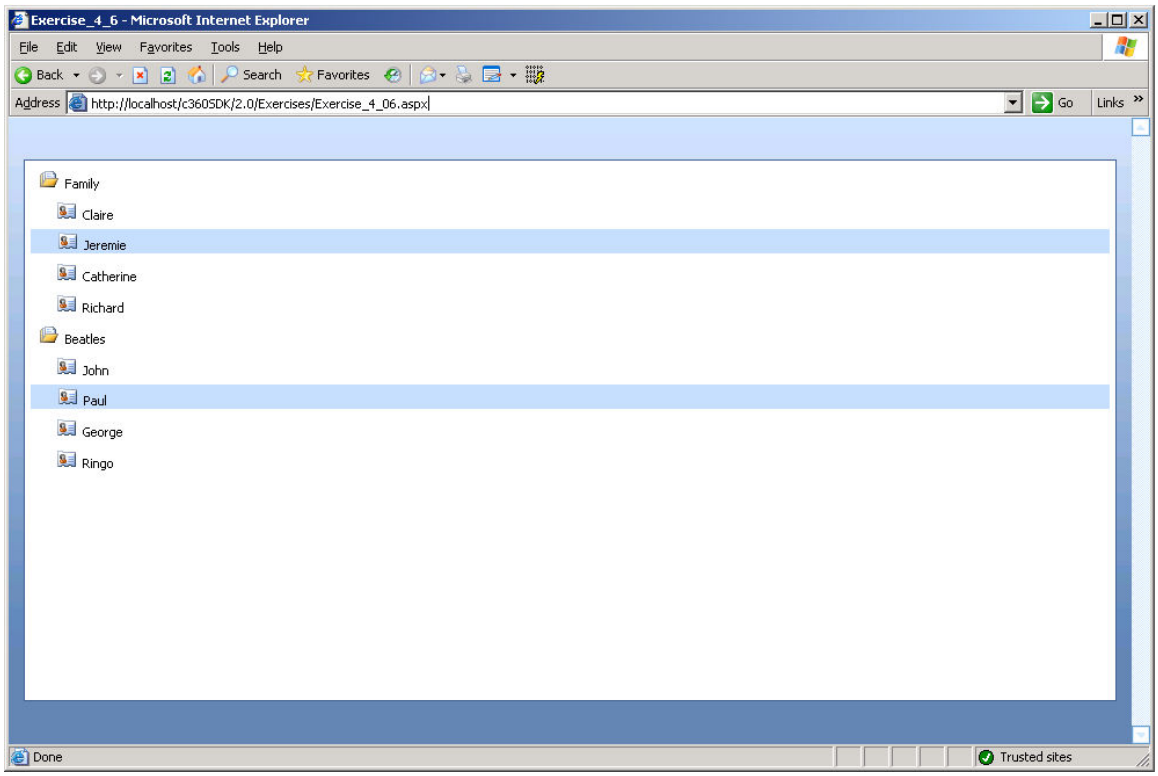
List (cont'd)

Exercise 4.6: List

Build a page deriving from Area which:

- Displays a list containing two groups of four items each
- Titles the first group, "Family". In it, list four members of your family, with one of them highlighted.
- Titles the second group, "The Beatles". In it, list the four members of the Beatles, with one of them highlighted.
- Has a CRM account icon for each group title
- Does not allow the user to click on the group titles
- Has a CRM contact icon for each group item
- Alerts the user the caption of the double-clicked item

Exercise 4.6 Desired Output



List (cont'd)

Exercise 4.6 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
	public override void BuildMenuArea() {
		c360.Toolkit.SDK.UI.IList objList = ControlFactory.CreateList();
		objList.ID = "TestList";
		objList.AllowMultiSelect = true;
		objList.Items.Add(ControlFactory.CreateListItem("Family", 1, "family", false, null, 0, false));
		objList.Items.Add(ControlFactory.CreateListItem("Claire", 2, "claire", true, null, 1, false));
		objList.Items.Add(ControlFactory.CreateListItem("Jeremie", 2, "jeremie", true, null, 1, true));
		objList.Items.Add(ControlFactory.CreateListItem("Catherine", 2, "catherine", true, null, 1, false));
		objList.Items.Add(ControlFactory.CreateListItem("Richard", 2, "richard", true, null, 1, false));
		objList.Items.Add(ControlFactory.CreateListItem("Beatles", 1, "beatles", false, null, 0, false));
		objList.Items.Add(ControlFactory.CreateListItem("John", 2, "john", true, null, 1, false));
		objList.Items.Add(ControlFactory.CreateListItem("Paul", 2, "paul", true, null, 1, true));
		objList.Items.Add(ControlFactory.CreateListItem("George", 2, "george", true, null, 1, false));
		objList.Items.Add(ControlFactory.CreateListItem("Ringo", 2, "ringo", true, null, 1, false));

		objList.DoubleClickEventHandler = "doubleClick";

		Instance.ContentArea.Controls.Add(objList.Control);
	}
</script>

<script language="JavaScript">
function doubleClick(){
	var strItemCaption = event.itemCaption;
	alert("You have double-clicked " + strItemCaption);
}
</script>
```

Lookup

Overview

The lookup control allows the user to look for a CRM record. Developers can specify which CRM entities to be searched, as well as from external data sources. This control can handle multiple entity types in the same lookup. The items in the lookup are defined in LookupItemCollection as LookupItems.

Lookup Server Side Properties

Server Side Property (type)	Description/Comments
AllowCreateNew (bool)	Indicates if the user is allowed to create a new record from the lookup window. `objMyLookup.AllowCreateNew = false;`
AllowedObjectTypes (ArrayList)	The list of entity types that can be looked up. `objMyLookup.AllowedObjectTypes.Add(112);`
AllowViewProperties (bool)	Indicates if the user is allowed to view record properties from the lookup window. `objMyLookup.AllowViewProperties = false;`
Disabled (bool)	Indicates if the control is disabled by default. `objMyLookup.Disabled = false;`

Filters (ILookupFilterCollection)	Collection of conditions that allows you to filter the records that can be looked up.
	
	Here is an example where the user can lookup contacts but he is restricted to search only for contacts related to a certain account:
	`ILookupFilter objFilter = ControlFactory.CreateLookupFilter();`
	`objFilter.AttributeName = "accountid";`
	`objFilter.EntityType = 2;`
	`objFilter.Operator = ConditionOperator.Equal;`
	`objFilter.Values.Add("1FD3F939-1C54-4622-876C-C00F4059CC7C");`
	
	`ILookup objMyLookup = ControlFactory.CreateLookup();`
	`objMyLookup.ID = "myLookup";`
	`objMyLookup.AllowedObjectTypes.Add(2);`
	`objMyLookup.Filters.Add(objFilter);`

ID (string)	The unique ID of the control. `objMyLookup.ID = “myLookup”;`

Lookup (cont'd)

Lookup Server Side Properties (cont'd)

Server Side Property (type)	Description/Comments
Items (LookupItemCollection)	The collection of items in the lookup selected items. `objMyLookup.Items.Add();`
LookupStyle (LookupStyles)	Defines the type of lookup control. Three possible values are single, multiple, or subject. `objMyLookup.LookupStyle = LookupStyles.Multiple;`
OnChangeEventHandler (LookupItemCollection)	The name of a JavaScript function that will be invoked when the user selects a different record. `objMyLookup.OnChangeEventHandler = "jsChangeHandler";`
LookupBrowse	The property is used to remove the top bar from the lookup that contains the lookup entity picklist `objMyLookup.LookupBrowse = True/False;`
AdditionalParameters (string)	Additional Parameters to be included in the url, if any The parameter will be send as a query string to the lookup page. `objMyLookup.AdditionalParameters = "Sample Additional Parameter"`

Lookup Server Side Methods

[There are no SDK-specific methods for this control]

Lookup Client Side Properties

Assuming that you have set the ID property to a unique value (let's say "MyLookup" for example), you can get access to the lookup object in your JavaScript code like this:

```
var objMyLookup = document.all.MyLookup;
```

After you have reference to this object, you have access to the following properties:

Client Side Property (type)	Description/Comments
lookupdisabled (bool)	Client side equivalent of the Disabled server-side property.
lookupstyle (string)	Client side equivalent of the LookupStyle server-side property.
lookupobjects (string)	Comma separated list of object types. Client side equivalent of the AllowedObjectTypes server-side property.
showNew (bool)	Client side equivalent of the AllowCreateNew server-side property.

Lookup (cont'd)

Lookup Client Side Properties (con't)

Client Side Property (type)	**Description/Comments**
showProp (bool)	Client side equivalent of the AllowViewProperties server-side property.

filter (string)	Contains the server-side Filters property in .NET serialized XML format. Here is an example of a filter on an integer field:
	`<LookupFilters>`
	`  <Values>`
	`  <anyType xsi:type="xsd:int">2</anyType>`
	`  <Values>`
	`  <EntityType>1</EntityType>`
	`  <AttributeName>accountratingcode</AttributeName>`
	`  <Operator>Equal</Operator>`
	`</LookupFilter>`
	
	The filter for a string is very similar except that the "xsd:int" is replaced with "xsd:string":
	`<LookupFilters>`
	`  <Values>`
	`  <anyType xsi:type="xsd:string">Bycicle Shop</anyType>`
	`  <Values>`
	`  <EntityType>1</EntityType>`
	`  <AttributeName>name</AttributeName>`
	`  <Operator>Equal</Operator>`
	`</LookupFilter>`

Lookup Client Side Methods

Client Side Method (return)	**Description/Comments**
Clear (void)	Clears the user selection.
Clear (void)	Returns an array where each element contains the CRM type of an item present in the list. Returns null if the list does not contain any items.
GetAllCaptions (array)	Returns an array where each element contains the caption of the items selected by the user. Returns null if the user hasn't made any selection.
GetAllTypes (array)	Returns an array where each element contains the CRM type of the items selected by the user. Returns null if the user hasn't made any selection.
GetAllValues (array)	Returns an array where each element contains the unique identifier of the items selected by the user. Returns null if the user hasn't made any selection.

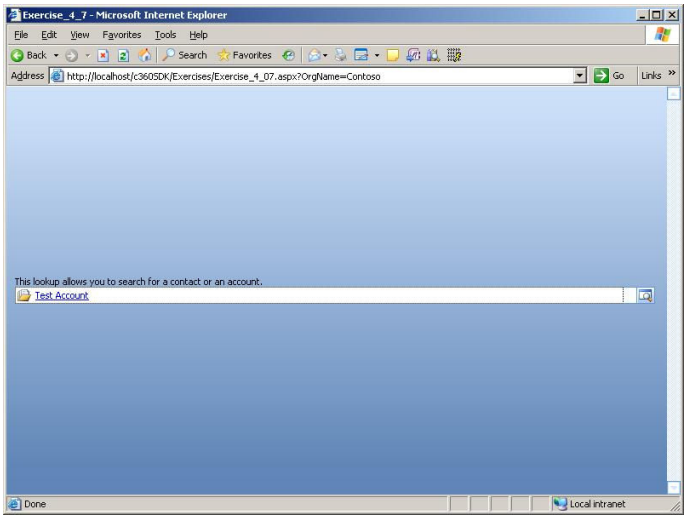
Lookup (cont'd)

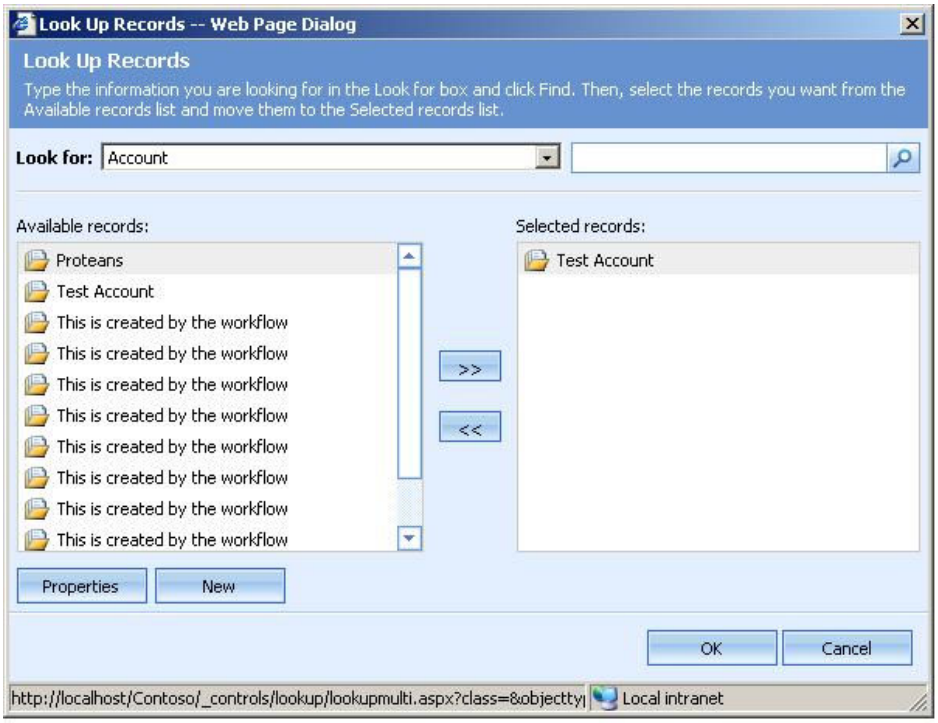
Exercise 4.7: Lookup

Create a page deriving from Area which:

- Contains a Lookup Control accepting objects of type Contact and Account
- The Lookup Control allows the user to view the properties of the record
- The Lookup Control allows the user to create a new record

Exercise 4.7 Desired Output





Lookup (cont'd)

Exercise 4.7 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
	public override void BuildMenuArea() {
		Instance.MenuArea.Controls.Add(new LiteralControl("Header"));
	}

	public override void BuildContentArea() {
		Instance.ContentArea.Controls.Add(new LiteralControl("This lookup allows you to search for a contact or an account."));
		ILookup objLookup = ControlFactory.CreateLookup();
		objLookup.AllowedObjectTypes.Add(1);
		objLookup.AllowedObjectTypes.Add(2);
		objLookup.AllowCreateNew = true;
		objLookup.AllowViewProperties = true;
		Instance.ContentArea.Controls.Add(objLookup.Control);
	}
</script>
```

Menu

Overview

The Menu control contains a menu bar with optional buttons and drop-down menus with optional sub-menus. It is more than just a simple menu, in fact, it would have been more descriptive to call it a “MenuBar”.

Menu Server Side Properties

Server Side Property (type)	**Description/Comments**
ID (string)	The unique ID of the control. `objMyMenu.ID = “myMenu”;`

Items (IMenuItemsCollection)	Collection of items that will render as either buttons or as pull-down menus. When an item in this collection has sub-items, it gets rendered as a pull-down menu. When it doesn’t have child items, it gets rendered as a button. This enables you to create a complex, multi-level menu.
	
	Here is an example of a menu item that will be rendered as a button on the menu bar:
	`IMenuItem objItem = ControlFactory.CreateMenuItem();`
	`objItem.Text = "Delete";`
	`objItem.Tooltip = "Click here to delete this record";`
	`objItem.IconPath = "/imgs/delete.gif";`
	`objItem.Action = "jsDeleteRecord();";`
	
	`IMenu objMenu = ControlFactory.CreateMenu();`
	`objMenu.Items.Add(objItem);`
	
	Here is an example of a menu item that will be rendered as a pull-down menu:
	`IMenuItem objItem = ControlFactory.CreateMenuItem();`
	`objItem.Items.Add(controlFactory.CreateMenuItem("Action 1", “/imgs/1.gif”);`
	`objItem.Items.Add(controlFactory.CreateMenuItem("Action 2", “/imgs/2.gif”);`
	`objItem.Items.Add(controlFactory.CreateMenuItem("Action 3", “/imgs/3.gif”);`
	
	`IMenu objMenu = ControlFactory.CreateMenu();`
	`objMenu.Items.Add(objItem);`

Style (MenuStyles)	Indicates the visual style of the menu. The three possible choices are “Normal”, “Toolbar” and “Flat”. `objMyMenu.Style = MenuStyles.Normal;`
Title (string)	This string will be displayed in yellow letters on the far right end of the menu bar. `objMyMenu.Title = “Menu title”;`

Menu (cont'd)

Menu Server Side Methods

[There are no SDK-specific methods for this control]

Menu Client Side Properties

[There are no SDK-specific properties for this control]

Menu Client Side Methods

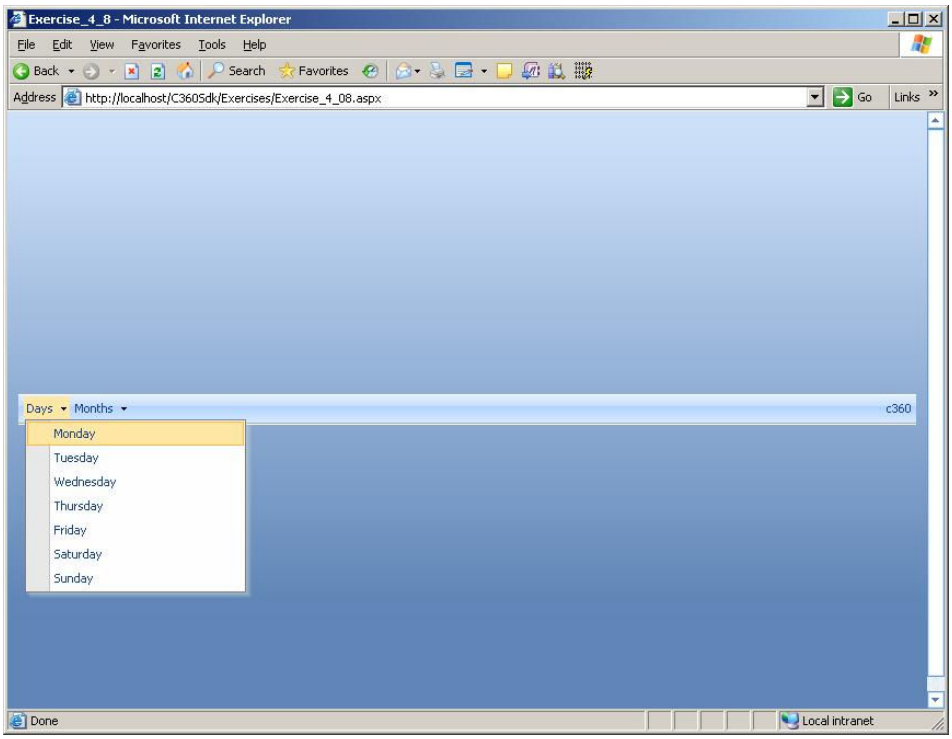
[There are no SDK-specific methods for this control]

Exercise 4.8: Menu

Build a page deriving from Menu which:

- Displays a menu bar with two pull-down menus
- Titles the first menu "Days"
- Titles the second menu "Months"
- Displays all the days of the week when the Days menu is clicked
- Displays all the months of the year when the Months menu is clicked
- Displays your name as the title of this menu bar
- Alerts the user of the day that is selected

Exercise 4.8 Desired Output



Menu (cont'd)

Exercise 4.8 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
	public override void BuildMenuArea() {
		IMenu objMenu = ControlFactory.CreateMenu();
		objMenu.ID = "TestMenu";
		objMenu.Title = "c360";

		//create the Months Menu
		IMenuItem daysMenu = ControlFactory.CreateMenuItem("Days");

		//add items to the Days Menu
		daysMenu.Items.Add(ControlFactory.CreateMenuItem("Monday", null, null, "selectionChange('Monday')"));
		daysMenu.Items.Add(ControlFactory.CreateMenuItem("Tuesday", null, null, "selectionChange('Tuesday')"));
		daysMenu.Items.Add(ControlFactory.CreateMenuItem("Wednesday", null, null, "selectionChange('Wednesday')"));
		daysMenu.Items.Add(ControlFactory.CreateMenuItem("Thursday", null, null, "selectionChange('Thursday')"));
		daysMenu.Items.Add(ControlFactory.CreateMenuItem("Friday", null, null, "selectionChange('Friday')"));
		daysMenu.Items.Add(ControlFactory.CreateMenuItem("Saturday", null, null, "selectionChange('Saturday')"));
		daysMenu.Items.Add(ControlFactory.CreateMenuItem("Sunday", null, null, "selectionChange('Sunday')"));

		//create the Months Menu
		IMenuItem monthsMenu = ControlFactory.CreateMenuItem("Months");

		//add items to the Months Menu
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("January"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("February"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("March"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("April"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("May"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("June"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("July"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("August"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("September"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("October"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("November"));
		monthsMenu.Items.Add(ControlFactory.CreateMenuItem("December"));

		//add the days menu and the months menu to the objMenu
		objMenu.Items.Add(daysMenu);
		objMenu.Items.Add(monthsMenu);

		Instance.ContentArea.Controls.Add(objMenu.Control);
	}
</script>
```

MultiPicklist

Overview

This control is similar in functionality to the Picklist control, but is intended to allow the user to select more than just one value. This control can either display static content, dynamic content, or Picklist values from CRM. This control is particularly useful in presenting Picklist values automatically from the CRM database, while allowing users to choose multiple values instead of just one.

MultiPicklist Server Side Properties

Server Side Property (type)	**Description/Comments**
AvailableItems (MultiPicklistItemCollection)	Collection of items available for the user to choose from.
Disabled (bool)	Indicates if the control is disabled by default. `objMyMultiPickList.Disabled = false;`
EntityName (string)	Defines the entity name of the object type to use in this control. Only necessary if you want this control to display a CRM Picklist field. `objMyMultiPickList.EntityName = "account";`
FieldName (string)	Defines the field name you'd like to display of the selected entity. Only necessary if you want this control to display a CRM Picklist field. This field MUST be of type CRM Picklist. `objMyMultiPickList.FieldName = "address1_addresstypecode";`
ID (string)	The unique ID of the control. `objMyMultiPickList.ID = "myMultiPickList";`
OnChangeEventHandler (string)	The name of a JavaScript function that will be invoked when the user selects a different record. `objMyMultiPickList.OnChangeEventHandler = "jsChangeHandler";`
SelectedItems (MultiPicklistItemCollection)	Collection of selected items in the MultiPicklist.

MultiPicklist (cont'd)

MultiPicklist Server Side Methods

[There are no SDK-specific methods for this control]

MultiPicklist Client Side Properties

Assuming that you have set the ID property to a unique value (let's say "MyMultiPickList" for example), you can get access to the multipicklist object in your JavaScript code like this:

```
var objMyMultiPickList = document.all.MyMultiPickList;
```

After you have reference to this object, you have access to the following properties:

Client Side Property (type)	**Description/Comments**
entityId (string)	Client side equivalent of the EntityName server-side property.
fieldName (string)	Client side equivalent of the FieldName server-side property.
picklistdisabled (bool)	Client side equivalent of the Disabled server-side property.

MultiPicklist Client Side Methods

Client Side Method (return)	**Description/Comments**
ClearPickList (void)	Clears the user selection.
GetAllCaptions (array)	Returns an array where each element contains the caption of the items selected by the user. Returns null if the user hasn't made any selection.
GetAllValues (array)	Returns an array where each element contains the unique identifier of the items selected by the user. Returns null if the user hasn't made any selection.
Picklist (void)	Pops the selection window.

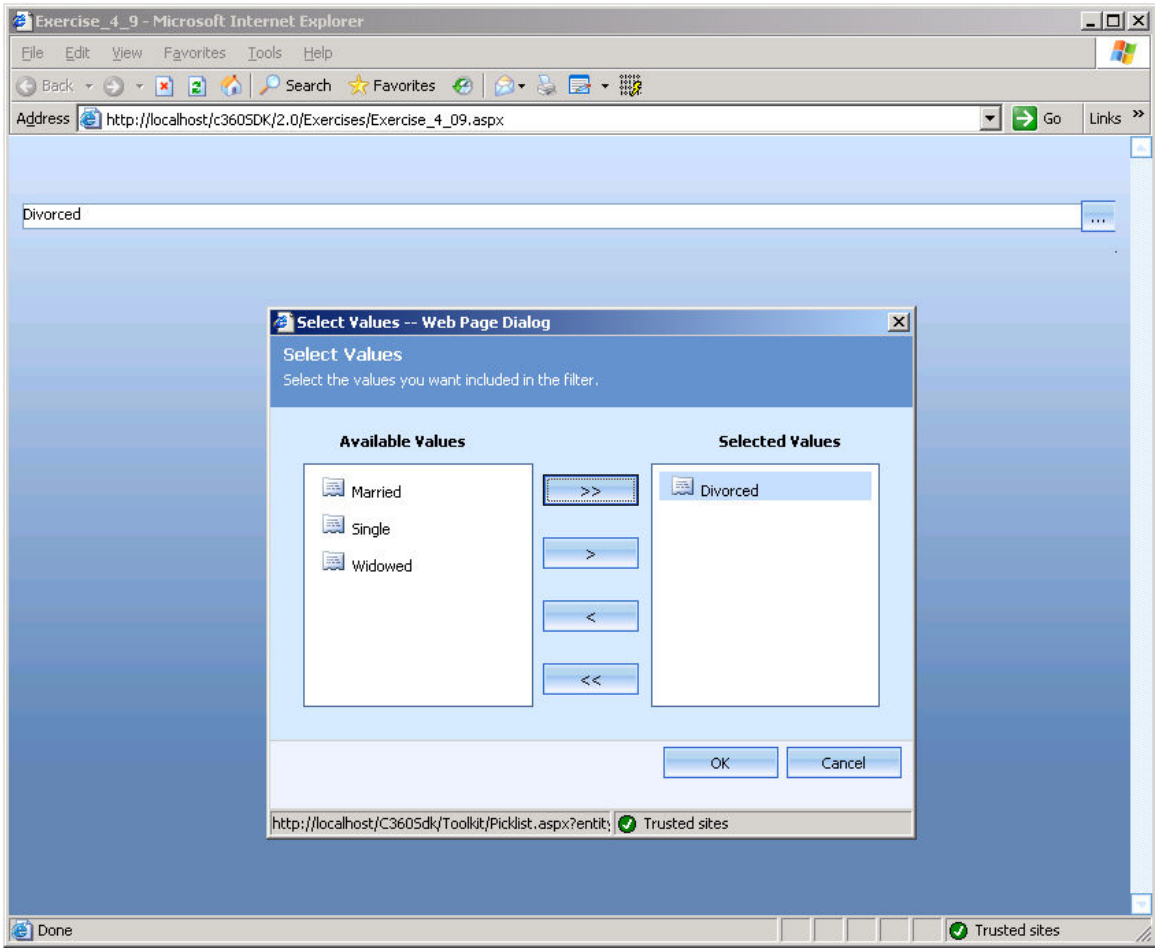
MultiPicklist (cont'd)

Exercise 4.9: MultiPicklist

Create a page deriving from Area which:

- The content area contains a MultiPicklist control that displays the familystatuscode picklist field for contacts.
- The user is alerted when the selection has changed.

Exercise 4.9 Desired Output



MultiPicklist (cont'd)

Exercise 4.9 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
        public override void BuildMenuArea() {
                Instance.MenuArea.Controls.Add(new LiteralControl("Header"));
        }

        public override void BuildContentArea() {
                IMultiPicklist objMultiPicklist = ControlFactory.CreateMultiPicklist();
                objMultiPicklist.EntityName = "contact";
                objMultiPicklist.FieldName = "familystatuscode";
                objMultiPicklist.OnChangeEventHandler = "PickListionChange";

                Instance.ContentArea.VerticalAlign = System.Web.UI.WebControls.VerticalAlign.Top;
                Instance.ContentArea.Controls.Add(objMultiPicklist.Control);
        }
</script>

<script language="javascript">
function PickListionChange(){
        alert("NOTICE: You have changed your selection");
}
</script>
```

Picklist

Overview

This control is the equivalent of an HTML select control, intended to display data for the user to pick one value. This control can either display static content, dynamic content, or Picklist values from CRM. This control is particularly useful in presenting Picklist values automatically from the CRM database.

Picklist Server Side Properties

Server Side Property (type)	**Description/Comments**
Disabled (bool)	Indicates if the control is disabled by default. `myPicklist.Disabled = false;`
EntityName (string)	Defines the entity name of the object type to use in this control. Only necessary if you want this control to display a CRM Picklist field. `objMyPickList.EntityName = "account";`
FieldName (string)	Defines the field name you'd like to display of the selected entity. Only necessary if you want this control to display a CRM Picklist field. This field MUST be of type CRM Picklist. `objMyPickList.FieldName = "address1_addresstypecode";`
ID (String)	The unique ID of the control. `objMyPickList.ID = "myPickList";`
Items (PicklistItemCollection)	Collection of available items. Each item can contain sub-items.
OnChangeEventHandler (string)	The name of a Javascript function to be invoked when the user selects a different record. `objMyPickList.OnChangeEventHandler = "jsChangeHandler";`
Selected (string)	The value of the selected item. `objMyPickList.Selected = "24";`

Picklist (cont'd)

Picklist Server Side Methods

[There are no SDK-specific methods for this control]

Picklist Client Side Properties

Assuming that you have set the ID property to a unique value (let's say "MyPickList" for example), you can get access to the picklist object in your JavaScript code like this:

```
var objMyPickList = document.all.MyPickList;
```

After you have reference to this object, you have access to the following properties:

Client Side Property (type)	**Description/Comments**
DataXml (string)	The data represented as a XML string.
DefaultValue (string)	The default value.
Disabled (bool)	Client-side equivalent of the Disabled server-side property.
IsDirty (bool)	Indicates is the selection has been modified by the user.
SelectedText (string)	The text of the selected option.

Picklist Client Side Methods

Client Side Method (return)	**Description/Comments**
SetFocus (void)	Sends the focus to this control.
AddOptionGroup (void)	Adds a "group" to the control. This method accepts the following parameters: the caption of this picklist item, the value of the picklist item and an optional string indicating the sorting option for the child items.
AddOption (void)	Adds an item to the picklist. This method accepts the following parameters: the caption of this picklist item, the value of the picklist item and an optional string indicating the "items group" that this item should be added to.
DeleteOption (void)	Removes an item from the list. This method accepts only one parameter which contains the value of the item to be removed.

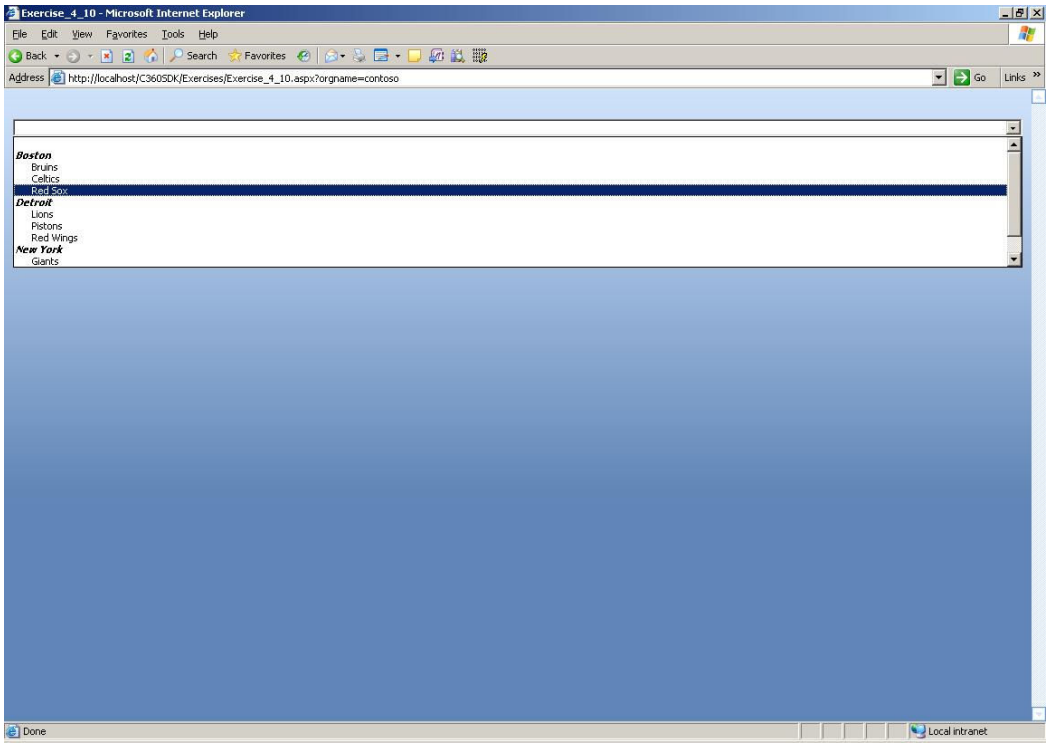
Picklist (cont'd)

Exercise 4.10: Picklist

Create a page deriving from Area which:

- The content area contains a Picklist control listing three cities that contain at least three major sport teams.
- Each city contains sub-items of the teams in that city.
- When the selection has changed, the user is alerted the value of the selection

Exercise 4.10 Desired Output



Picklist (cont'd)

Exercise 4.10: Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
	public override void BuildMenuArea() {
		Instance.MenuArea.Controls.Add(new LiteralControl("Header"));
	}

	public override void BuildContentArea() {
		IPicklist PickListMenu = ControlFactory.CreatePicklist();
		PickListMenu.ID = "PickListTeam";
		PickListMenu.Items.Add(ControlFactory.CreatePicklistItem());
		PickListMenu.OnChangeEventHandler = "PickListionChange";

		IPicklistItem itmBoston = ControlFactory.CreatePicklistItem("Boston","Boston");
		itmBoston.Items.Add(ControlFactory.CreatePicklistItem("Red Sox","Boston Red Sox"));
		itmBoston.Items.Add(ControlFactory.CreatePicklistItem("Bruins","Boston Bruins"));
		itmBoston.Items.Add(ControlFactory.CreatePicklistItem("Celtics","Boston Celtics"));
		PickListMenu.Items.Add(itmBoston);

		IPicklistItem itmNY = ControlFactory.CreatePicklistItem("New York","New York");
		itmNY.Items.Add(ControlFactory.CreatePicklistItem("Giants","NY Giants"));
		itmNY.Items.Add(ControlFactory.CreatePicklistItem("Yankees","NY Yankees"));
		itmNY.Items.Add(ControlFactory.CreatePicklistItem("Knicks","NY Knicks"));
		PickListMenu.Items.Add(itmNY);

		IPicklistItem itmDetroit = ControlFactory.CreatePicklistItem("Detroit","Detroit");
		itmDetroit.Items.Add(ControlFactory.CreatePicklistItem("Pistons","Detroit Pistons"));
		itmDetroit.Items.Add(ControlFactory.CreatePicklistItem("Lions","Detroit Lions"));
		itmDetroit.Items.Add(ControlFactory.CreatePicklistItem("Red Wings","Detroit Red Wings"));
		PickListMenu.Items.Add(itmDetroit);

		Instance.ContentArea.VerticalAlign = System.Web.UI.WebControls.VerticalAlign.Top;
		Instance.ContentArea.Controls.Add(PickListMenu.Control);
	}
</script>

<script language="javascript">
function PickListionChange(){
	var strItemCaption = document.all.PickListTeam.value;
	alert("You have PickListed " + strItemCaption);
}
</script>
```

Radio

Overview

The Radio control creates a radio button list that can be displayed vertically or horizontally. Note: Microsoft CRM uses the radio control with a “Yes” and a “No” options to represent Boolean fields.

Radio Server Side Properties

Server Side Property (type)	**Description/Comments**
Alignment (enum called “Alignments”)	The two possible values in this enum are "Vertical" and "Horizontal" (default: Vertical). This property allows you to specify if you want the items to be listed all on the same row (Horizontal) or each on a separate row (Vertical). `myRadio.Alignment = Alignments.Horizontal;`
Disabled (bool)	Indicates if the control is disabled by default. `myRadio.Disabled = false;`
ID (String)	The unique ID of the control. `objMyRadio.ID = “myRadio”;`

Items (IRadioItemCollection)	This is a collection of items that will be available to the user . You can programmatically add additional items like this:
	`objMyRadio.Items.Add(new ControlFacotry.CreateRadioItem("Yes", "y"));`
	`objMyRadio.Items.Add(new ControlFacotry.CreateRadioItem("No", "n"));`

OnChangeEventHandler (string)	Name of a JavaScript function that will be called when the user changes the selection. This is useful if you want to be notified of the selection change. The JavaScript function you specify will not be passed any parameters when fired. `objMyRadio.OnchangeEventHandler = “jsChangeHandler”;`
Selected (string)	Value of the item selected by default. `objMyRadio.Selected = “y”;`
Title (string)	The caption displayed above the items. `objMyRadio.Title = “Send Marketing information”;`

Radio (cont'd)

Radio Server Side Methods

[There are no SDK-specific methods for this control]

Radio Client Side Properties

[There are no SDK-specific properties for this control]

Radio Client Side Methods

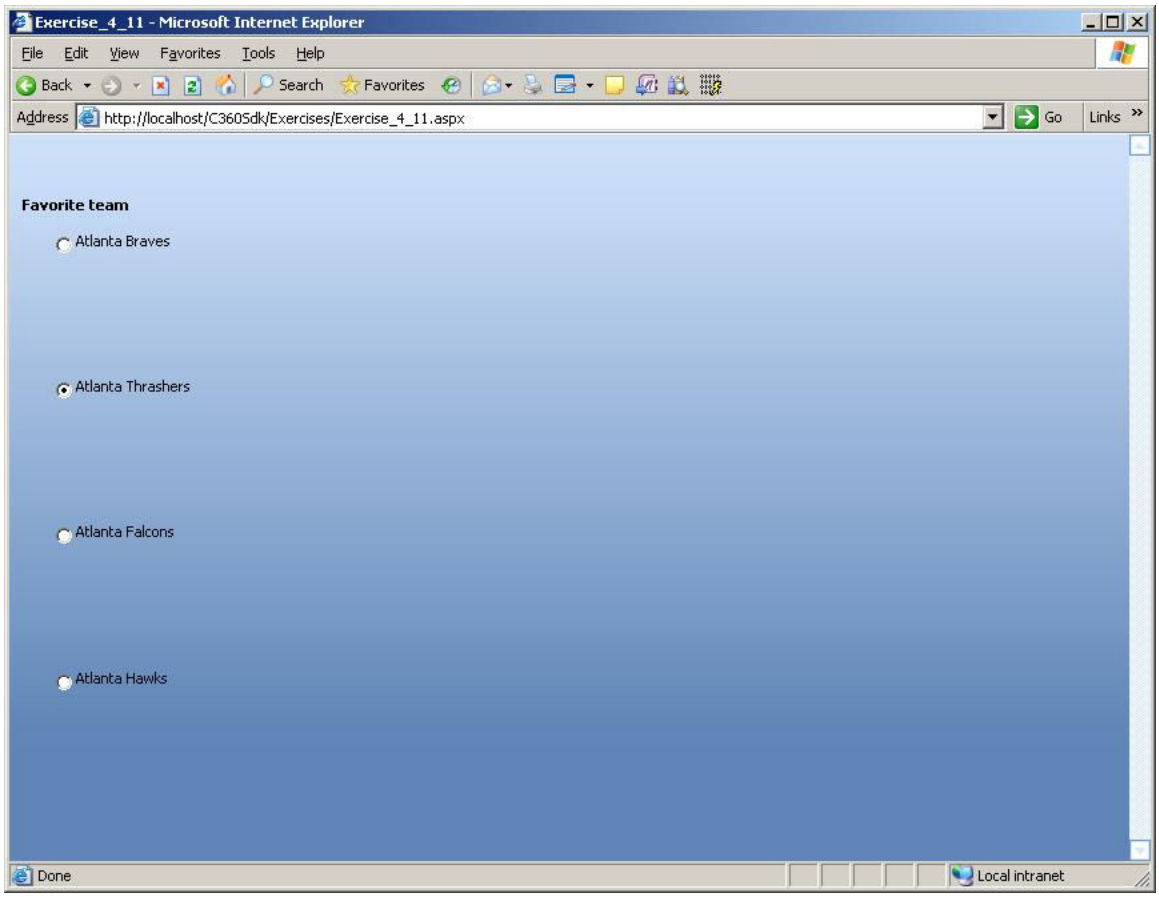
[There are no SDK-specific methods for this control]

Exercise 4.11: Radio Control

Build a page deriving from Area which:

- Displays a radio control with the title of "Favorite Team"
- Lists the four Atlanta professional sports teams as radio items
- Defaults to the "Atlanta Thrashers"

Exercise 4.11 Desired Output



Radio (cont'd)

Exercise 4.11 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
	public override void BuildMenuArea() {
		IRadio objRadio = ControlFactory.CreateRadio();
		objRadio.ID = "TestRadio";
		objRadio.Title = "Favorite team";
		objRadio.Selected = "Thrashers";
		objRadio.Items.Add(ControlFactory.CreateRadioItem("Atlanta Braves", "Braves"));
		objRadio.Items.Add(ControlFactory.CreateRadioItem("Atlanta Thrashers", "Thrashers"));
		objRadio.Items.Add(ControlFactory.CreateRadioItem("Atlanta Falcons", "Braves"));
		objRadio.Items.Add(ControlFactory.CreateRadioItem("Atlanta Hawks", "Hawks"));

		Instance.ContentArea.Controls.Add(objRadio.Control);
	}
</script>
```

TabBar

Overview

The TabBar control is an interface for controlling information presented in tabbed areas that can be viewed/hidden by clicking on the tab.

TabBar Server Side Properties

Server Side Property (type)	Description/Comments
ClientSideAfterChangeHandler (string)	The name of a Javascript function to be called after the focus is sent to another tab. `myTabBar.ClientSideAfterChangeHandler = "jsAfterChangeHandler";`
ClientSideBeforeChangeHandler (string)	The name of a Javascript function to be called before the focus is set to another tab. `myTabBar.ClientSideBeforeChangeHandler = "jsBeforeChangeHandler";`
ID (String)	The unique ID of the control. `myTabBar.ID = "myTabBar";`

Items (ITabItemCollection)	Collection of tabs. When adding a tab, you must specify the following 3 properties:
	• The caption of the tab
	• The unique identifier of the tab
	• A boolean to indicate if this tab must be highlighted by default
	You can add Tabs on the TabBar by adding to the Items collection like in the following example:
	`TabBar objTabBar = ControlFactory.CreateTabBar();`
	`objTabBar.Items.Add(ControlFactory.CreateTabItem("General", "tab0", true));`
	`objTabBar.Items.Add(ControlFactory.CreateTabItem("Advanced", "tab1", false));`
	You can add content to each tab by adding web controls to each tab's Controls collection like in the following example:
	`objTabBar.Items[0].Controls.Add(new LiteralControl("This is a test"));`

TabBar Server Side Methods

[There are no SDK-specific methods for this control]

TabBar Client Side Properties

[There are no SDK-specific properties for this control]

TabBar Client Side Methods

[There are no SDK-specific methods for this control]

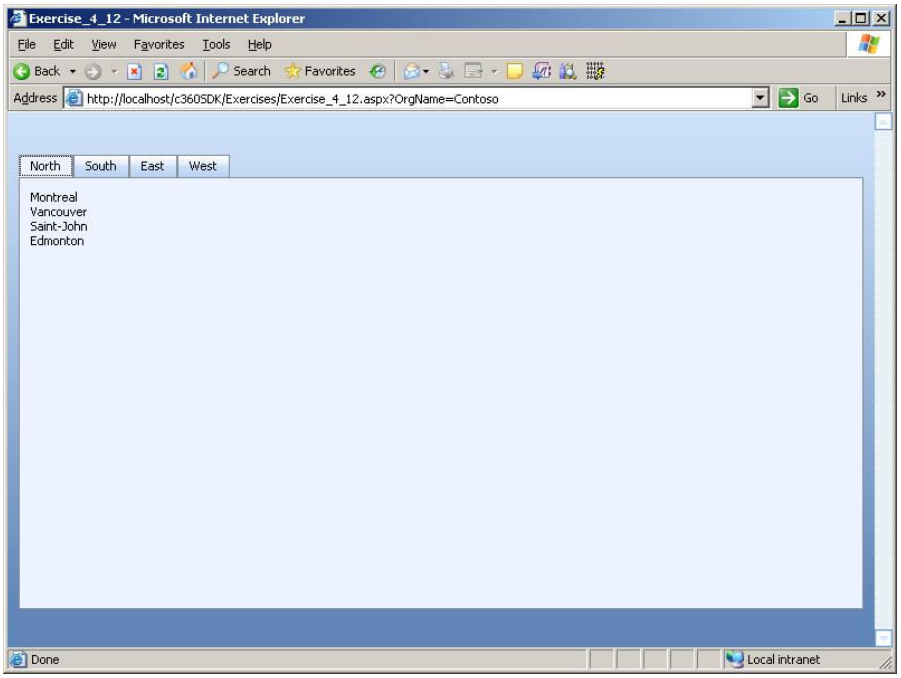
TabBar (cont'd)

Exercise 4.12: TabBar

Build a page deriving from Area which:

- Displays a four-item tab bar
- Has tabs: North, South, East, West
- Displays the name of four Canadian cities in the North tab
- Displays the name of four South-American Countries in the South tab
- Displays the name of four European countries in the East tab
- Displays the name of four US states in the West tab

Exercise 4.12 Desired Output



TabBar (cont'd)

Exercise 4.12 Solution

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
public override void BuildMenuArea() {
		ITabBar objTabBar = ControlFactory.CreateTabBar();
		objTabBar.ID = "TabBar";

		ITabItem objNorthTab = ControlFactory.CreateTabItem("North", "tab0", true);
		objNorthTab.Controls.Add(new LiteralControl("Montreal<br>"));
		objNorthTab.Controls.Add(new LiteralControl("Vancouver<br>"));
		objNorthTab.Controls.Add(new LiteralControl("Saint-John<br>"));
		objNorthTab.Controls.Add(new LiteralControl("Edmonton<br>"));

		ITabItem objSouthTab = ControlFactory.CreateTabItem("South", "tab1", false);
		objSouthTab.Controls.Add(new LiteralControl("Costa-Rica<br>"));
		objSouthTab.Controls.Add(new LiteralControl("Brazil<br>"));
		objSouthTab.Controls.Add(new LiteralControl("Chile<br>"));
		objSouthTab.Controls.Add(new LiteralControl("Argentina<br>"));

		ITabItem objEastTab = ControlFactory.CreateTabItem("East", "tab2", false);
		objEastTab.Controls.Add(new LiteralControl("France<br>"));
		objEastTab.Controls.Add(new LiteralControl("Germany<br>"));
		objEastTab.Controls.Add(new LiteralControl("Italie<br>"));
		objEastTab.Controls.Add(new LiteralControl("Spain<br>"));

		ITabItem objWestTab = ControlFactory.CreateTabItem("West", "tab3", false);
		objWestTab.Controls.Add(new LiteralControl("Maine<br>"));
		objWestTab.Controls.Add(new LiteralControl("Georgia<br>"));
		objWestTab.Controls.Add(new LiteralControl("Texas<br>"));
		objWestTab.Controls.Add(new LiteralControl("California<br>"));

		objTabBar.Items.Add(objNorthTab);
		objTabBar.Items.Add(objSouthTab);
		objTabBar.Items.Add(objEastTab);
		objTabBar.Items.Add(objWestTab);

		Instance.ContentArea.Controls.Add(objTabBar.Control);
	}
</script>
```

TreeView

Overview

The TreeView control allows data to be presented hierarchically. This control can get its data from a TreeViewDynamicContent Page, and allows for asynchronous data fetching when a user expands a node allowing for much better user experience.

TreeView Server Side Properties

Server Side Property (type)	Description/Comments
AllowMultiSelect (bool)	Indicates whether the user can highlight more than one record of not. `myTreeView.AllowMultiSelect = false;`
ClickEventHandler (string)	The name of a JavaScript function to be called when the user clicks on an item. `myTreeView.ClickEventHandler = "jsClickHandler";`
DoubleClickEventHandler (string)	The name of a JavaScript function to be called when the user double-clicks on an item. `myTreeView.DoubleClickEventHandler = "jsDoubleClickHandler";`
FetchDataPage (string)	URL to a custom page where data is asynchronously fetched. `myTreeView.FetchDataPage = "TreeData.aspx";`
ID (String)	The unique ID of the control. `myTreeView.ID = "myTreeView";`
Items (TreeViewItemsCollection)	Collection of items in the tree.
ShowEntityIcon (bool)	Indicates if the CRM entity icon is to be displayed next to each item in the tree. `myTree.ShowentityIcon = true;`

TreeView Server Side Methods

[There are no SDK-specific methods for this control]

TreeView Client Side Properties

[There are no SDK-specific properties for this control]

TreeView (cont'd)

TreeView Client Side Methods

[There are no SDK-specific methods for this control]

Exercise 4.13: TreeView

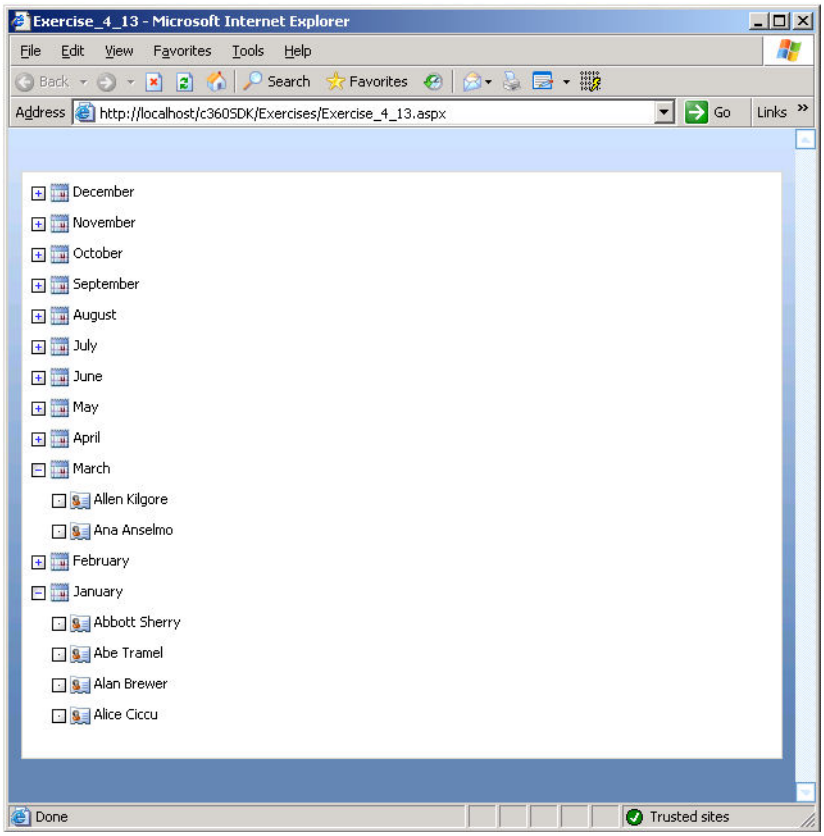
Create a page deriving from Area which:

- The content area contains a TreeView control
- The TreeView control lists the twelve months of the year
  - Create an icon to use for the months
  - Each month should contain dynamic
- Allows the user to select only one object at a time
- Has a TreeViewDynamicContent page defined to dynamically fetch the data when a user expands one of these months

Create a page deriving from TreeViewDynamicContent which:

- Dynamically add TreeViewItems under the month node of contacts who have a birthday during that month (build a QueryExpression that utilizes the base.ObjId that is inherited from the TreeViewDynamicContent class)
  - Display the contact's full name

Exercise 4.13 Desired Output



TreeView (cont'd)

Exercise 4.13 Solution

TreeView Interface

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.AreaPage" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>

<script runat="server">
	public override void BuildMenuArea() {
		Instance.MenuArea.Controls.Add(new LiteralControl("Header"));
	}

	public override void BuildContentArea() {
		ITreeView objTree = ControlFactory.CreateTreeView();
		objTree.FetchDataPage = "Exercise_4_13_treeviewcontent.aspx";
		objTree.ID = "tree";
		objTree.AllowMultiSelect = false;

		objTree.Items.Add(ControlFactory.CreateTreeViewItem("December", "12", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("November", "11", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("October", "10", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("September", "9", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("August", "8", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("July", "7", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("June", "6", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("May", "5", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("April", "4", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("March", "3", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("February", "2", "month_icon.JPG", false, false, true));
		objTree.Items.Add(ControlFactory.CreateTreeViewItem("January", "1", "month_icon.JPG", false, false, true));

		Instance.ContentArea.VerticalAlign = System.Web.UI.WebControls.VerticalAlign.Top;
		Instance.ContentArea.Controls.Add(objTree.Control);
	}
</script>
```

TreeView (cont'd)

TreeView Content

```
<%@ Page Inherits="c360.Toolkit.SDK.Pages.TreeViewDynamicContent" Language="C#" %>
<%@ Import Namespace="c360.Toolkit.SDK.UI" %>
<%@ Import Namespace="c360.Toolkit.CRM" %>

<script runat="server">
        public override void FetchData() {

                int thisMonth = System.Int32.Parse(base.ObjId); //for using this page with a treeview control
                string beginDate = thisMonth.ToString() + "/01/2005";
                int endDay = DateTime.DaysInMonth(2005,thisMonth);
                string endDate = thisMonth.ToString() + "/" + endDay.ToString() + "/2005";

                ConditionExpression objConditions = new ConditionExpression();
                objConditions.AttributeName = "birthdate";
                objConditions.Operator = ConditionOperator.Between;
                objConditions.Values = new object[]{ beginDate, endDate };

                // Define what we need to fetch
                FilterExpression objFilter = new FilterExpression();
                objFilter.FilterOperator = LogicalOperator.And;
                objFilter.Conditions = new ConditionExpression[] {objConditions};

                // Get the contact
                QueryExpression objQuery = new QueryExpression();
                objQuery.EntityName = EntityName.contact.ToString();
                objQuery.ColumnSet = new AllColumns();
                objQuery.Criteria = objFilter;

                RetrieveMultipleRequest objRequest = new RetrieveMultipleRequest();
                objRequest.Query = objQuery;

                RetrieveMultipleResponse objContacts = (RetrieveMultipleResponse) CrmService.Execute(objRequest);

                for (int i = 0; i < objContacts.BusinessEntityCollection.BusinessEntities.Length; i++) {
                        contact thisContact = (contact) objContacts.BusinessEntityCollection.BusinessEntities[i];
                        string name = ((contact)objContacts.BusinessEntityCollection.BusinessEntities[i]).fullname;
                        string contactid = ((contact)objContacts.BusinessEntityCollection.BusinessEntities[i]).contactid.ToString();
                        AddTreeViewItemToReturn(name, 2, contactid, null, false, false, false);
                }
        }
</script>
```

5. Microsoft CRM SDK

The classes and objects mentioned here will greatly aid developers in building custom Microsoft CRM functionality. Our pages and controls will save time constructing pages to look, act, and feel like existing CRM components and therefore the majority of your development efforts will be with the logic and handling of data in conjunction with our pages and controls. However, mimicking the look and feel is often not enough. You also need to interact with the CRM system to extract data and/or update existing data.

Microsoft provides several a very extensive API that allows you interact with the CRM system. Our SDK provides a subset of this API via three objects: the "CrmService", "MetadataService" and "CrmDiscoveryService"

The CrmService object gives you access to a few of the methods provided by the Microsoft CRM API such as "ExecuteFetchXml", "DeleteEntity" and "UpdateEntity" while the MetadataService object gives you access to methods that allow you to retrieve "metadata" about the CRM system whereas CrmDiscoveryService provides you to access to methods that allows you to retrieve "CrmServiceUrl" and "CrmMetaDataServiceUrl" Keep in mind that not all the functionality of Microsoft's API is replicated via these objects and there may be some situations where you still need to reference Microsoft's web services instead of using the two convenient objects we provide.

The Microsoft Business Solutions CRM V4.0 SDK contains a wealth of resources, including code samples, which are designed to help you build powerful vertical applications using the Microsoft CRM platform. The Microsoft CRM SDK includes documentation that provides a wide range of instructive and practical information.

It will be important to familiarize yourself with much of the CRM 4.0 SDK as it will be essential when working with CRM data. The Microsoft CRM 4.0 SDK introduces some new methods (vs. Earlier Versions) of retrieving CRM data: CrmService, MetadataService and CrmDiscoveryService. The properties and methods provided in these classes are available through the Microsoft Web Services WSDL.

Referencing the MSCRM Services from your Project

In order to leverage these services, you must add references to the WSDL files to your .NET project. The WSDL files are located here:

http://<crm server name>/mscrmservices/2007/crmservice.asmx?wsdl
http://<crm server name>/mscrmservices/2007/metadataservice.asmx?wsdl

CrmService

Overview

CrmService is a property of all c360 web pages that allows you to invoke various methods to fetch, update and delete records from the CRM database. This Web Service is called to from the c360 SDK Base page, and can be referenced from other pages by calling:

```
Instance.CrmService
```

CrmService Methods

Method Name (return)	Description/Comments

ConvertFetchXmlToQueryExpression (QueryExpression)	Converts a fetch Xml string into a QueryExpression object.
	
	`string fetchXml = "<fetch mapping='logical'><entity name='account'>";`
	`fetchXml += "<all-attributes/>";`
	`fetchXml += "<filter type='and'>";`
	`fetchXml += "<condition attribute='address1_country' operator='eq' value='U.S.'/>";`
	`fetchXml += "</filter>";`
	`fetchXml += "</entity></fetch>";`
	
	`QueryExpression queryExpression =`
	`Instance.CrmService.ConvertFetchXmlToQueryExpression(fetchXml);`

ConvertQueryExpressionToFetch (string)	Converts a QueryExpression object into a fetch Xml string.
	
	`ConditionExpression objConditions = new ConditionExpression();`
	`objConditions.AttributeName = "birthdate";`
	`objConditions.Operator = ConditionOperator.Between;`
	`objConditions.Values = new object[] { "01/01/2005", "01/31/2005" };`
	
	`FilterExpression objFilter = new FilterExpression();`
	`objFilter.FilterOperator = LogicalOperator.And;`
	`objFilter.Conditions = new ConditionExpression[] { objConditions };`
	
	`QueryExpression objQuery = new QueryExpression();`
	`objQuery.EntityName = EntityName.contact.ToString();`
	`objQuery.ColumnSet = new AllColumns();`
	`objQuery.Criteria = objFilter;`
	
	`string fetchXml = Instance.CrmService.ConvertQueryExpressionToFetch(objQuery);`

CreateEntity (Guid)	Creates a new record in the CRM database.
	
	`contact contact = new contact();`
	`contact.firstname = "Jesper";`
	`contact.lastname = "Aaberg";`
	
	`Guid contactGuid = Instance.CrmService.CreateEntity(contact);`

CrmService Methods (cont'd)

Method Name (return)	Description/Comments

DeleteEntity (Guid)	Deletes a record in the CRM database.
	`Guid contactGuid = new Guid("50694E79-2029-4449-9197-6FBC0BF9023F");`
	`Instance.CrmService.DeleteEntity(EntityName.contact.ToString(), contactGuid);`

Execute (Response)	Executes a web service ""request and returns a "response". Please see the Microsoft CRM SDK documentation for more details.

ExecuteFetchXml (string)	Executes a fetch Xml query and returns the result as a XML string.
	`string fetchXml = "<fetch mapping='logical'><entity name='account'>";`
	`fetchXml += "<all-attributes/>";`
	`fetchXml += "<filter type='and'>";`
	`fetchXml += "<condition attribute='address1_country' operator='eq' value='U.S.'/>";`
	`fetchXml += "</filter>";`
	`fetchXml += "</entity></fetch>";`
	
	`string resultset = Instance.CrmService.ExecuteFetchXml(fetchXml);`

RetrieveBusinessEntity (BusinessEntity)	Retrieves a record from the CRM database and returns it as a Business Entity.
	`ColumnSet cols = new ColumnSet();`
	`cols.Attributes = new string[] { "fullname" };`
	`Guid contactGuid = new Guid("50694E79-2029-4449-9197-6FBC0BF9023F");`
	
	`contact contact =`
	`(contact)Instance.CrmService.RetrieveBusinessEntity(EntityName.contact.ToString(), contactGuid, cols);`

RetrieveDynamicEntity (DynamicEntity)	Retrieves a record from the CRM database and returns it as a Dynamic Entity.
	`ColumnSet cols = new ColumnSet();`
	`cols.Attributes = new string[] { "fullname" };`
	`Guid contactGuid = new Guid("50694E79-2029-4449-9197-6FBC0BF9023F");`
	
	`DynamicEntity entity = Instance.CrmService.RetrieveDynamicEntity(EntityName.contact.ToString(), contactGuid, cols);`

CrmService Methods (cont'd)

Method Name (return)	Description/Comments

RetrieveMultipleBusinessEntities (BusinessEntityCollection)	Retrieves records from the CRM database based a query.
	`ConditionExpression objConditions = new ConditionExpression();`
	`objConditions.AttributeName = "birthdate";`
	`objConditions.Operator = ConditionOperator.Between;`
	`objConditions.Values = new object[] { "01/01/2005", "01/31/2005" };`
	
	`FilterExpression objFilter = new FilterExpression();`
	`objFilter.FilterOperator = LogicalOperator.And;`
	`objFilter.Conditions = new ConditionExpression[] { objConditions };`
	
	`QueryExpression objQuery = new QueryExpression();`
	`objQuery.EntityName = EntityName.contact.ToString();`
	`objQuery.ColumnSet = new AllColumns();`
	`objQuery.Criteria = objFilter;`
	
	`BusinessEntityCollection contacts =`
	`Instance.CrmService.RetrieveMultipleBusinessEntities(objQuery);`

RetrieveMultipleDynamicEntities (BusinessEntityCollection)	Retrieves records from the CRM database based a query.
	`ConditionExpression objConditions = new ConditionExpression();`
	`objConditions.AttributeName = "birthdate";`
	`objConditions.Operator = ConditionOperator.Between;`
	`objConditions.Values = new object[] { "01/01/2005", "01/31/2005" };`
	
	`FilterExpression objFilter = new FilterExpression();`
	`objFilter.FilterOperator = LogicalOperator.And;`
	`objFilter.Conditions = new ConditionExpression[] { objConditions };`
	
	`QueryExpression objQuery = new QueryExpression();`
	`objQuery.EntityName = EntityName.contact.ToString();`
	`objQuery.ColumnSet = new AllColumns();`
	`objQuery.Criteria = objFilter;`
	
	`BusinessEntityCollection contacts =`
	`Instance.CrmService.RetrieveMultipleDynamicEntities(objQuery);`

UpdateEntity (void)	Deletes a record in the CRM database.
	`contact contact = new contact();`
	`contact.address1_line1 = "34 Market St.";`
	`contact.contactid = new Key();`
	`contact.contactid.Value = new Guid("4D507FFE-ED25-447B-80DE-00AE3EB18B84");`
	
	`Instance.CrmService.UpdateEntity(contact);`

CrmService Methods (cont'd)

Method Name (return)	Description/Comments

String **ConvertQueryExpressionToFetch**(QueryExpression queryExpression)	Converts a QueryExpression to FetchXml
	ConditionExpression objConditions = new ConditionExpression();
	objConditions.AttributeName = "birthdate";
	objConditions.Operator = ConditionOperator.Between;
	objConditions.Values = new object[] { "01/01/2005", "01/31/2005" };
	
	FilterExpression objFilter = new FilterExpression();
	objFilter.FilterOperator = LogicalOperator.And;
	objFilter.Conditions = new ConditionExpression[] { objConditions };
	
	QueryExpression objQuery = new QueryExpression();
	objQuery.EntityName = EntityName.contact.ToString();
	objQuery.ColumnSet = new AllColumns();
	objQuery.Criteria = objFilter;
	
	string fetchXml = Instance.CrmService. ConvertQueryExpressionToFetch (objQuery);

QueryExpression **ConvertFetchXmlToQueryExpression**(string fetchXml);	Converts FetchXml to a Query Expression string fetchXml = "<fetch mapping='logical'><entity name='account'>"; fetchXml += "<all-attributes/>"; fetchXml += "<filter type='and'>"; fetchXml += "<condition attribute='address1_country' operator='eq' value='U.S.'/>"; fetchXml += "</filter>"; fetchXml += "</entity></fetch>"; QueryExpression queryExpression = Instance.CrmService. ConvertFetchXmlToQueryExpression (fetchXml);
Response **Execute**(Request request, bool throwException);	Executes CRM Request. Set 2nd parameter to True, if the exception policy is to be ignored and forcefully exception should be thrown in case of any error while executing the request. Please see the Microsoft CRM SDK documentation for more details Response response = Instance.CrmService.Execute(Request, true);
int **Timeout** { set; }	To set the timeout of the service

MetadataService

Overview

MetadataService is a property of all c360 web pages that allows you to invoke various methods to fetch metadata information about your CRM system. The metadata contains information the entities, attributes, and relationship definitions for your installation of Microsoft CRM, including your customizations.

This MetadataService Web Service contains the methods you need to discover all of the entities in the system. This data is read-only, and there are no methods for manipulating the metadata. You must use the customization tools in the application to add, for example, entities, attributes, and customer forms. The MetadataService is optimized for performance, containing separate methods with various flags to limit the data retrieved from the server.

This Web Service is called to from the c360 SDK Base page, and can be referenced from other pages by calling:

```
Instance.MetadataService
```

This Web service contains the methods you need to discover all of the entities in the system. It can also be used to build a client side cache. This data is read-only; there are no methods for manipulating the metadata. You must use the customization tools in the application to add, for example, entities, attributes, and customer forms.

The metadata service can be used to perform a variety of functions. With this service, you can:

- Check if a certain custom entity already exists in the system.
- Retrieve the metadata for a specific entity.
- Retrieve the metadata for a specific entity field.
- Retrieve the metadata for all entities.

MetadataService (cont'd)

MetadataService Methods

Method Name (return)	Description/Comments

DoesEntityExist (bool)	Indicates if a given entity exists in the current CRM system.
	`bool customEntity1Exist = Instance.MetadataService.DoesEntityExist(10001);`
	`bool customEntity2Exist = Instance.MetadataService.DoesEntityExist("new_CustomEntity");`

EntityNameToType (int)	Returns the integer value representing a given entity name. `string entityName = Instance.MetadataService. EntityNameToType(10001, true);`
EntityTypeToName (string)	Returns the name of an entity for a given integer value. `int entityType = Instance.MetadataService. EntityTypeToName("new_CustomEntity", true);`
RetrieveAttributeMetadata (AttributeMetadata)	Returns the metadata for a given entity field. `AttributeMetadata metaData = Instance.MetadataService. RetrieveAttributeMetadata("new_CustomEntity", "new_Field1");`
RetrieveEntities (EntityMetadata[])	Returns metadata for all entities. `EntityMetadata[] metaData = Instance.MetadataService. RetrieveEntities();`
RetrieveEntityMetadata (EntityMetadata)	Returns metadata for a given entity. `EntityMetadata metaData = Instance.MetadataService. RetrieveEntityMetadata("new_CustomEntity");`
ClearEntityMetadata	To Clear the EntityMetadataHash and EntityMetadataArray. `Instance.MetadataService.ClearEntityMetadata()`

CrmDiscoveryService

Overview

You'll use this service to 'discover' information about the different organizations on a server (deployment).

The CrmDiscoveryService Web service is a global installation-level service that allows the caller to determine the correct organization and URL for their needs. Because Microsoft Dynamics CRM 4.0 is a multi-tenant environment, a single Microsoft Dynamics CRM server can be hosting multiple business organizations. Because each Microsoft Dynamics CRM server may be handling a Web Service Method call for a different organization every time, the Web services must be notified of the target organization that a user is intending to reach.

The CrmDiscoveryService Web service supports a Web service request to return a list of organizations that the specified user belongs to and the URL endpoint addresses of each organization that hosts Microsoft Dynamics CRM server.

CrmDiscoveryService Methods

Method Name (return)	Description/Comments
SortedList<String, OrganizationDetail> OrganizationDetails	Returns the sorted list of organization details for the current user. `Instance.DiscoveryService. OrganizationDetails;`
String GetCrmServiceUrl(string organizationName**);**	This method fetches the CrmService url from the CrmDiscoveryService provided if there is no specific override in the c360.Config. You have to pass the OrganizationName to get the related CrmServiceUrl `String CrmServiceUrl = Instance.DiscoveryService.GetCrmServiceUrl( orgName );`
String GetCrmMetaDataServiceUrl(string organizationName**);**	This method fetches the CrmMetaDataService url from the CrmDiscoveryService provided if there is no specific override in the c360.Config. You have to pass the OrganizationName to get the related CrmMetaDataService url. `String MetadataServiceUrl = Instance.DiscoveryService.GetCrmMetaDataServiceUrl( orgName );`

CrmDiscoveryService (cont'd)

Method Name (return)	Description/Comments
String GetCrmUrl(string organizationName);	This method fetches the Crm url from the CrmDiscoveryService provided if there is no specific override in the c360.Config. You have to pass the OrganizationName to get the related Crm url. `String crmUrl = Instance.DiscoveryService.GetCrmUrl(orgName );`
(RetrievePolicyResponse) **discoveryService.Execute**(policyRequest);	Returns an instance of a Response. You must cast the return value of the Execute method to the specific instance of the Response that corresponds to the Request Parameter. `RetrievePolicyResponse policyResponse = (RetrievePolicyResponse)discoveryService.Execute(policyRequest);`

6. GridAction

Overview

Purpose / Background

The Action base page class is useful to develop a custom action that will be executed on each selected record in a grid. This page allows you to design a screen to capture additional information from the user and to design the action to be executed; all the other details are taken care of for you. This includes updating a progress bar to inform the user of the progress of the action.

How are records processed?

The concept behind this class is that the developer writes a method which treats one record at a time in the derived class. The base class takes care of repeatedly calling this method for each record and updating the progress bar between each call. When all records are treated, the action dialog is closed and the grid where this action originated is refreshed.

The concept of treating records one at a time may not be suitable to every situation. If you find yourself needing to write an action where all records must be processed at once, this may not be the best solution for you. You may want to consider using the "Actions" collection on the grid to add your action instead of the "CustomActions" because basic actions allow you to execute any JavaScript when the user clicks on the menu item.

How do I implement the GridAction class?

To build an interface where the user can specify additional information, proceed as if you are building a regular page by simply overriding the "BuildContent" method. You also need to write a client-side JavaScript function (this function is called the "PreSaveHandler") where you aggregate all the user input into an xml string; finally, you override the "SaveData" method.

The SaveData method is the method that will be called for each record and it's where you must process each record.

The following three (3) parameters will be passed to this function:

1. The unique identifier of the record being processed
2. The entity type of the record
3. The xml string containing the information you collected in the JavaScript PreSavedHandler.

Optionally, you can write a client-side validation handler which allows you to stop the action before records are processed if you are not satisfied with the user input. You can also write a success handler and an error handler.

Overview (cont'd)

GridAction Server Side Properties

Server Side Property (type)	**Description/Comments**
RecordCount (int)	The number of records that the user selected in the grid.
ClientSideValidationHandler (string)	Name of a client-side function where the user input is validated. This function must return a boolean indicating whether the action can be executed or not.
ClientSidePreSaveHandler (string)	Name of a client-side function where the user input is collected into an xml string and the result returned as a string. Please note that you can simply return an empty string if there is not user input to collect.
ClientSideSaveErrorHandler (string)	Name of a client-side function that will be invoked in case of an error. The following three (3) parameters will be passed to this function: · The description of the error · The unique identifier of the record · The entity type of the record
ClientSideSaveSuccessHandler (string)	Name of a client-side function that will be invoked after each record is successfully processed.
DialogWidth (int)	The width of the dialog in pixels.
DialogHeight (int)	The height of the dialog in pixels.

GridAction Server Side Methods

Server Side Method (return)	**Description/Comments**
SaveData (string)	You must override this method to process the selected records. If the record is successfully treated, you must return an empty string. However, if an error occurred, you must return the description of that error.
AddOkButton (void)	Call this method if you want to display the "Ok" button in the footer area of the dialog window.
AddCancelButton (void)	Call this method if you want to display the "Cancel" button in the footer area of the dialog window.

Overview (cont'd)

GridAction Client Side Properties

Client Side Property (type)	**Description/Comments**
_selectedRecords (array)	Array containing a two-element array for each selected record. Each two-element array contains the record's unique identifier and entity type.
_recordCount (int)	The total number of records to be processed.
_currentRecord (int)	The position of the record being processed. Before the process is started, this value is set to zero. Changing this value as the records are being processed to a greater value will cause some records to be skipped; changing it to a smaller value will cause some records to be processed again.
_saveString (string)	The xml data that will be passed to the SaveData method with each record.

GridAction Client Side Methods

Client Side Method (return)	**Description/Comments**
InitializeProgressBar() (void)	Displays the progress bar and sets the progress to zero percent.
UpdateProgressBar() (void)	Updates the progress bar after each record is processed.
FinalizeProgressBar() (void)	Called when all records are processed.
ExecuteActionOnRecord() (void)	Asynchronously executes the action on the current record.
btnCancel_onClick() (void)	Cancels the action and closes the dialog window. Please note that if the process has been started, the user is prompted to confirm if he really wants to cancel the action.

7. Miscellaneous

Current User

The "Instance" object of any c360 page has a property called "CurrentUser" which gives you access to various properties and methods about the CRM user who is using the custom solution you have created.

This CurrentUser normally refers to the identity of the end user who is currently interacting with your custom application but also keep in mind that it could refer to another user if you have enabled impersonation.

CurrentUser Properties

Property (type)	**Description/Comments**
IsAdministrator (bool)	Indicates if the user is member of the "System Administrator" role. Please note that you can specify the name or GUID of the role you want to be considered the administrators role. Refer to the "Post-Installation" section for details. `bool isAdministrator = Instance.CurrentUser.IsAdministrator;`
LocalDateTime (DateTime)	Current date and time expressed in the user's timezone. `datetime dt = Instance.CurrentUser.LocalDateTime;`
Organization (organization)	The organization that the user belongs to. `string orgName = Instance.CurrentUser.Organization.name;`
Settings (usersettings)	The user's CRM settings. `string homePage = Instance.CurrentUser.Settings.homepagearea;`
User (systemuser)	The CRM systemuser. `string myName = Instance.CurrentUser.User.fullname;`
UserCredentials (SortedList)	The windows Credentials of the current or impersonated user. `ICredentials myCredentials = Instance.CurrentUser.UserCredentials;`
WindowsPrincipalName (string)	The full windows principal name. `string myWindowsName = Instance.CurrentUser.WindowsPrincipalName;`

Checking Role membership

In addition to the properties listed in the previous section, the “CurrentUser” object also allows you to verify if the user is member of a certain role. This can help you determine if certain functionality should be available to a user or not.

CurrentUser Method

Server Side Method (return)	**Description/Comments**
IsMemberOf (bool)	Indicates if the current user is member of the specified role. You can pass either the name of a role or the GUID of the role in question. `bool isVicePresident = Instance.CurrentUser.IsMemberOf(“VP of Sales”);`

User Preferences

In Microsoft CRM, users have a certain number of personal settings they can update. An example of a user setting is the number of records per page the user wants to see in a grid. However there is an important limitation associated with user settings: developers cannot add new ones for their own purpose.

Our SDK solves this problem by allowing you to save and read User Preferences (we call them 'preferences' to prevent any confusion with the 'settings'). The preferences are loaded and cached in memory and therefore are very fast to access. However, because they are in memory, I recommended that you use it only for a few small values and not for storing large amount of data. Just be reasonable!

There are three methods that you can use to read and save preferences:

Server Side Method (return)	**Description/Comments**
GetPreference (string)	Returns the value of the specified preference. The user preferences XML file is loaded in the cache if it's not already loaded. `string myColor = Instance.CurrentUser.GetPreference("FavoriteColor");`
RemovePreference (void)	Remove a preference from the user preference XML file. `Instance.CurrentUser.RemovePreference("FavoriteColor");`
SetPreference (void)	Updates the value of a preference in the user preference XML file. `Instance.CurrentUser.SetPreference("FavoriteColor", "Green");`

Behind the scenes, the preferences are saved in small XML files in a folder called 'UserPreferences' off of the root of your solution. Please note that since each user will be updating their preference XML file, you need to make sure that you grant your users the ability to read and write in this folder.

8. Samples

All Controls

Overview

This very simple sample includes one of every c360 web control on one page. This sample is great to get familiar with all the controls included in the SDK.

This sample demonstrates the following:

- Edit page to enforce the CRM look and feel but hide the menu bar and button bar
- Use a 'LeftNav' control to mimic the Microsoft left navigation menu
- Use the 'TabBar' control to mimic the Microsoft CRM tabs on the edit page
- Insert one of every type of c360 control in one of the tabs

Installation

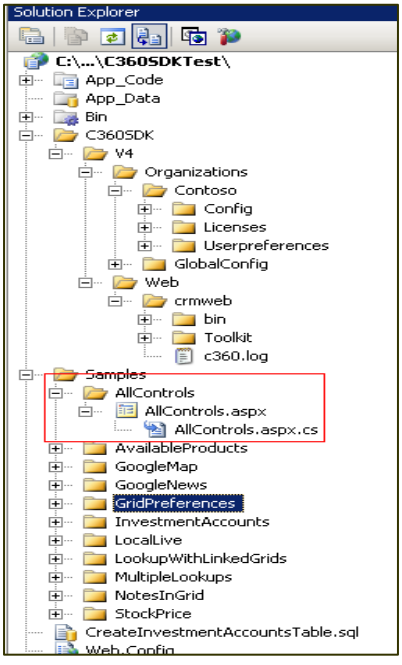
Step 1 – Edit ISV.config

This sample must be added as a new link under a "top level" menu (i.e.: a pull down menu on the top menu bar at the top of the main CRM window). Therefore we need to edit the "CustomMenus" section in Microsoft CRM's ISV.config file like so:

```
  <Menu Title="Testing c360 SDK">
    <MenuItem Title="All Controls"
Url="http://localhost:5517/c360SdkTest/Samples/AllControls/AllControls.aspx?orgname=<OrganizationName>" WinMode="0"
Client="Web,OutlookWorkstationClient" AvailableOffline="false" />
  </Menu>
```

Note: you will need to adjust the URL in the above sample to match your environment

Step 2 – Copy files



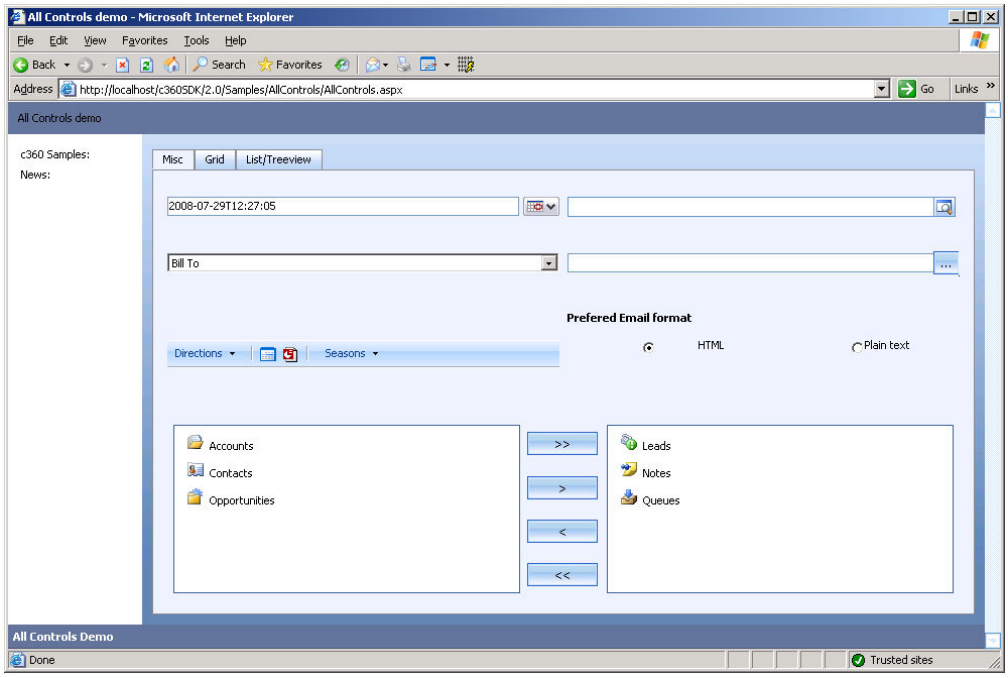
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "AllControls".

In this new folder, copy the ASPX page and its corresponding code behind file:

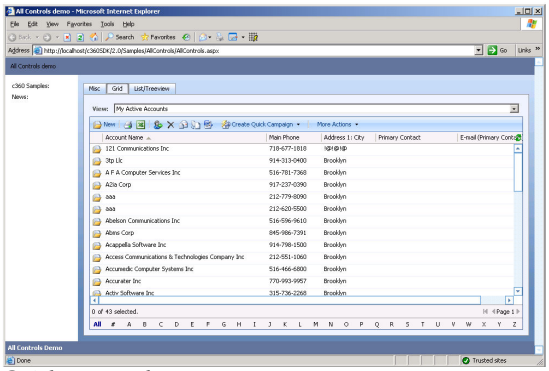
- AllControls.aspx
- AllControls.aspx.cs

All Controls (con't)

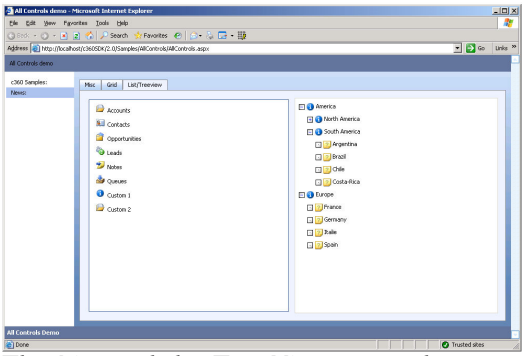
Screenshots



Main tab showing most of the c360 controls



Grid control



The List and the TreeView controls

Available Products

Overview

The scenario that was used for this sample is that a user wanted to simplify adding products to an Opportunity.

This sample demonstrates the following:

- List all products in a grid using a system view
- Add a “CustomAction” to a grid (the “delete” action)
- Use an “ActionPage” to implement the custom action
- Implement a custom “DoubleClickHandler” in a grid
- Use an “Edit” page to create new records and update existing ones

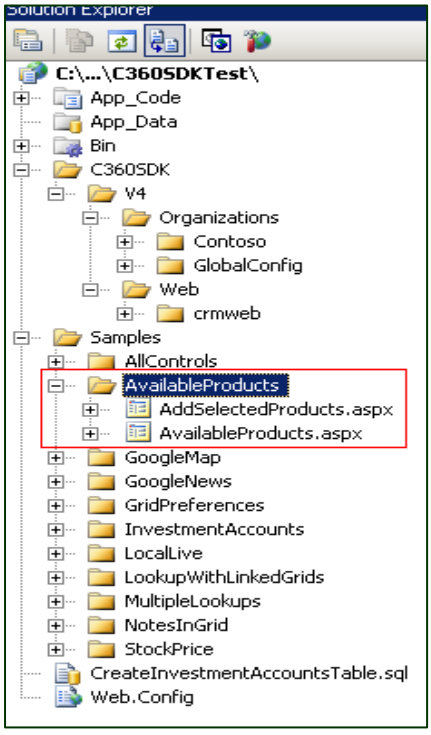
Installation

Step 1 – Edit ISV.config

This sample must be added as a new link on the left navigation bar of the Microsoft CRM opportunity edit screen. Therefore we need to edit the “opportunity” entity section in Microsoft CRM’s ISV.config file like so:

```
  <Entity name="opportunity">
    <NavBar ValidForCreate="0" ValidForUpdate="1">
     <NavBarItem Icon="/_imgs/ico_18_debug.gif" Title="Available Products"
Url="http://localhost:1465/c360SdkTest/Samples/AvailableProducts/AvailableProducts.aspx?orgname=<OrganizationName>"
Id="availableProducts" />
    </NavBar>
```

Note: you will need to adjust the URL in the above sample to match your environment.



Step 2 – Copy files

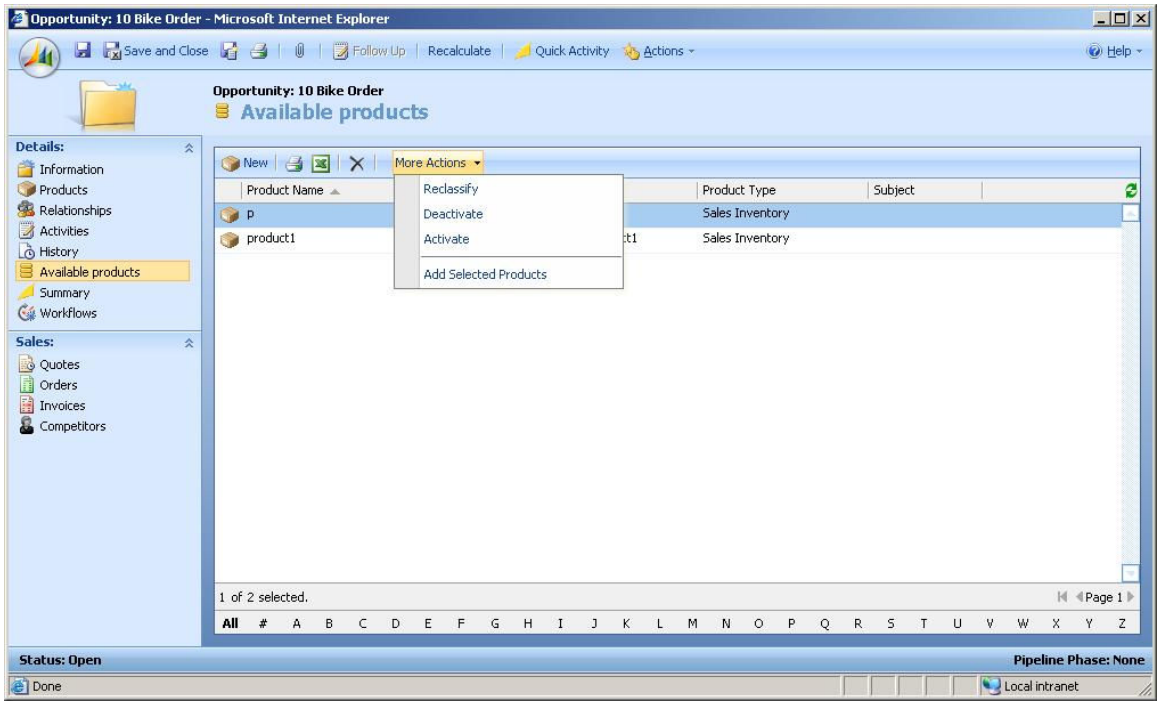
In your Visual Studio 2005 project, make sure you have a folder called “Samples” and create a sub-folder called “AvailableProducts”.

In this new folder, copy the two ASPX pages and their corresponding code behind files:

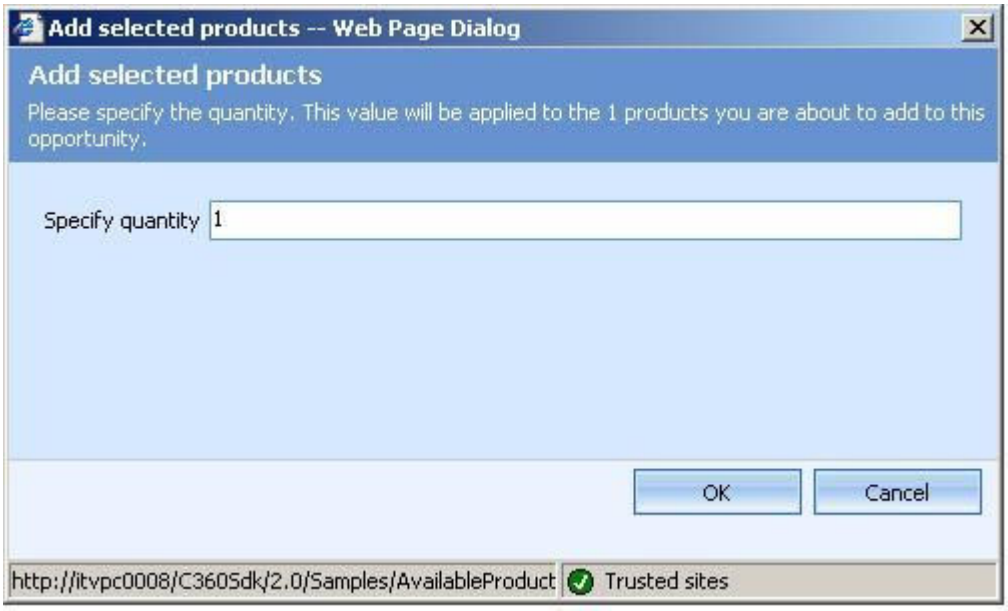
- AddSelectedProducts.aspx
- AddSelectedProducts.aspx.cs
- AvailableProducts.aspx
- AvailableProducts.aspx.cs

Available Products (con't)

Screenshots



List of available products (notice the custom action at the bottom of the menu)



Adding selected products to an opportunity

Google Map

Overview

The goal of this sample is to display a map of the area where an account or contact is located. Also, in the case of an order, we want to display directions to get to the delivery destination. (Please note that the starting address for the directions has be set to c360 Solutions; main office in Atlanta, but feel free to change it to your own address).

This sample demonstrates the following:

- Fetch data about the current account or order
- Redirect the user to Google map and pass the information expected by Google in order to be able to display a map

Installation

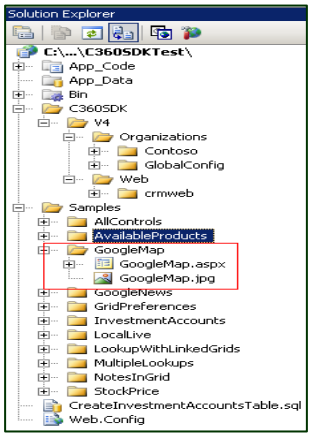
Step 1 – Edit ISV.config

This sample must be added as a new link on the left navigation bar of the Microsoft CRM account edit screen and the order edit screen as well. Therefore we need to edit the "account" and the "salesorder" entity sections in Microsoft CRM's ISV.config file like so:

```
  <Entity name="account">
    <NavBar ValidForCreate="0" ValidForUpdate="1">
      <NavBarItem Icon="http://localhost:4589/c360SdkTest/Samples/GoogleMap/GoogleMap.jpg" Title="Google map"
Url="http://localhost:4589/c360SdkTest/Samples/GoogleMap/GoogleMap.aspx?orgname=<OrganizationName>" Id="googleMap" />
    </NavBar>
  </Entity>
  <Entity name="salesorder">
    <NavBar ValidForCreate="0" ValidForUpdate="1">
      <NavBarItem Icon="http://localhost:4589/c360SdkTest/Sample/GoogleMap/GoogleMap.jpg" Title="Google map"
Url="http://localhost:4589/c360SdkTest/Samples/GoogleMap/GoogleMap.aspx?orgname=<OrganizationName>" Id="googleMap" />
    </NavBar>
  </Entity>
```

Note: you will need to adjust the URL in the above sample to match your environment.

Step 2 – Copy files



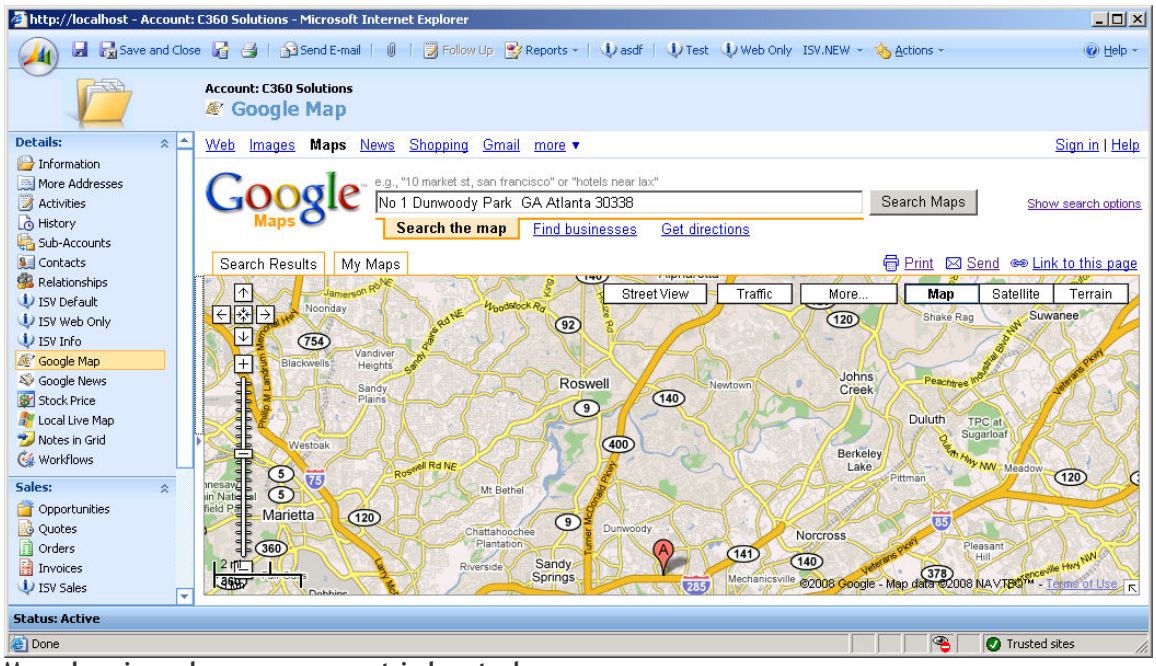
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "GoogleMap".

In this new folder, copy the image, the ASPX page and its corresponding code behind file:

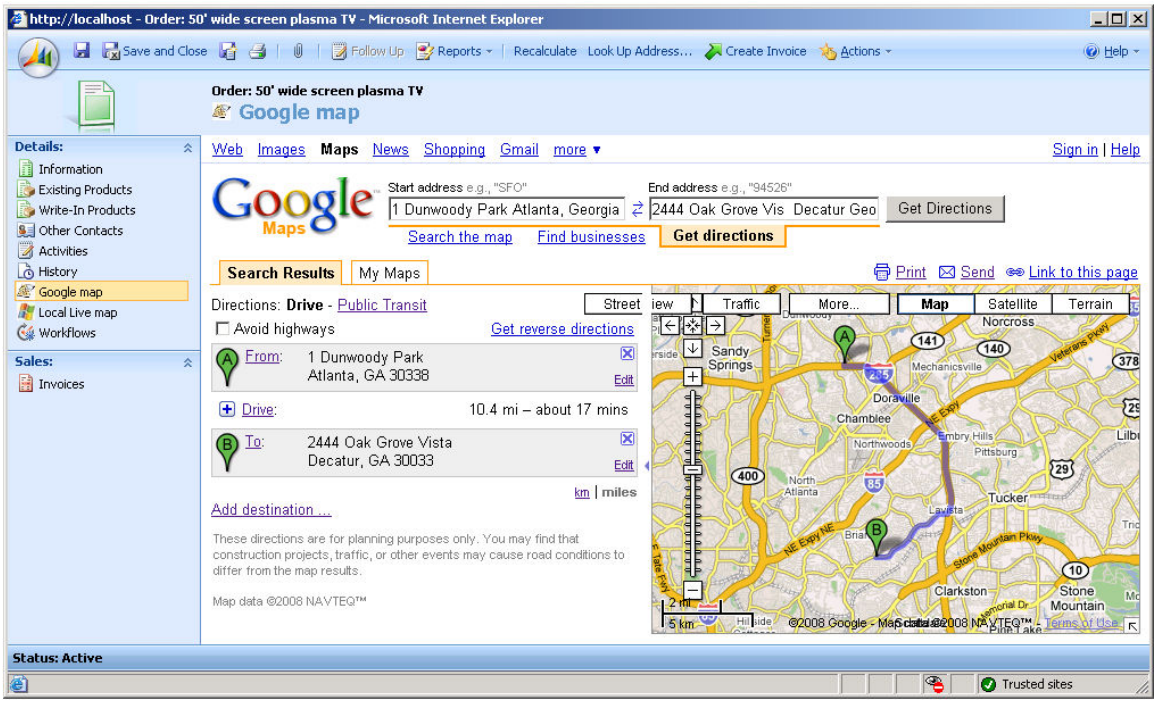
- GoogleMap.jpg
- GoogleMap.aspx
- GoogleMap.aspx.cs

Google Map (con't)

Screenshots



Map showing where an account is located



Map showing directions where an order must be delivered

Google News

Overview

The goal of this sample is to fetch and display news articles related to an account.

This sample demonstrates the following:

- Fetch data about the current account
- Fetch news articles from Google's RSS news feed
- Display the articles

Installation

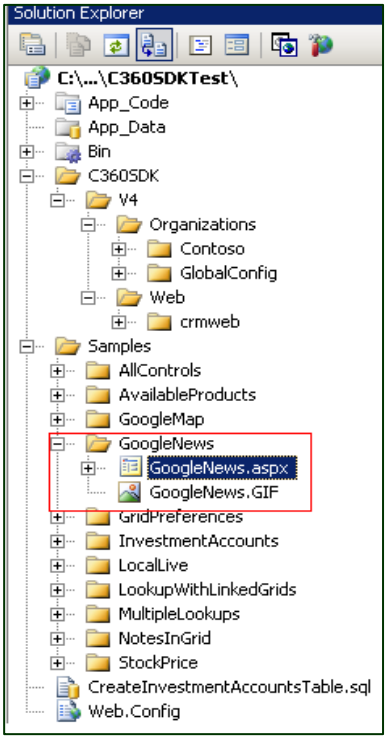
Step 1 – Edit ISV.config

This sample must be added as a new link on the left navigation bar of the Microsoft CRM account edit screen. Therefore we need to edit the "account" entity section in Microsoft CRM's ISV.config file like so:

```
  <Entity name="account">
    <NavBar ValidForCreate="0" ValidForUpdate="1">
      <NavBarItem Icon="http://localhost:4589/c360SdkTest/Samples/GoogleNews/GoogleNews.gif" Title="Google news"
Url="http://localhost:4589/c360SdkTest/Samples/GoogleNews/GoogleNews.aspx?orgname=<OrganizationName>" Id="googleNews"
/>
    </NavBar>
  </Entity>
```

Note: you will need to adjust the URL in the above sample to match your environment.

Step 2 – Copy files



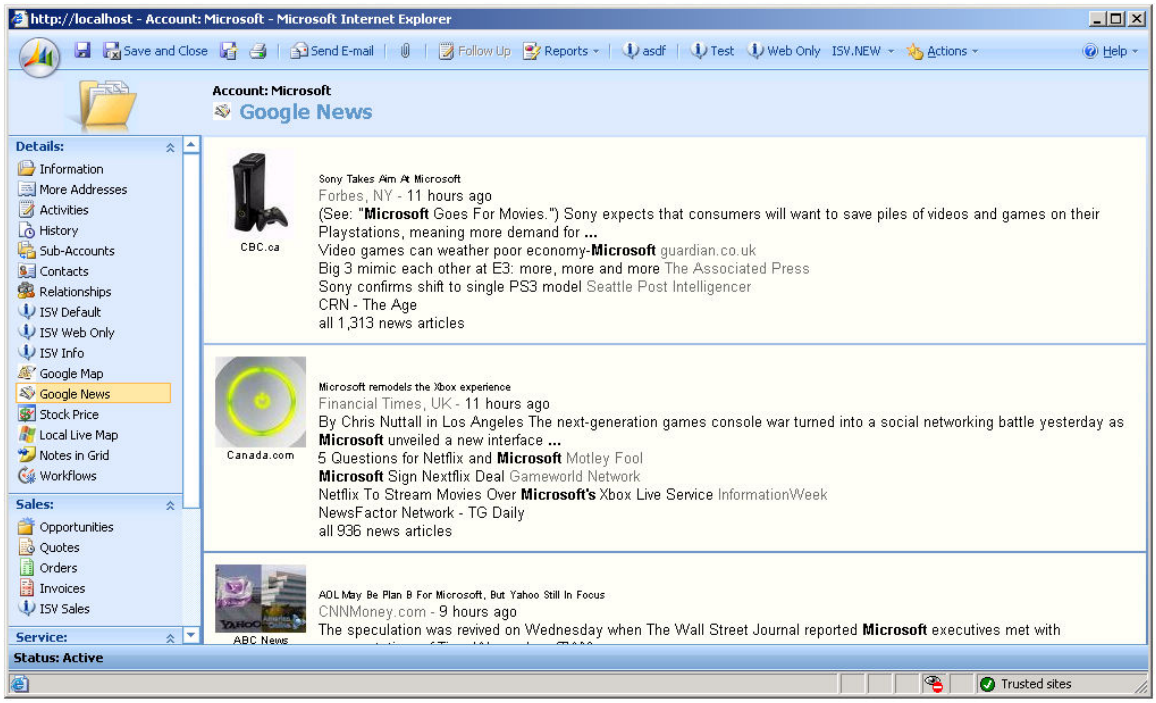
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "GoogleNews".

In this new folder, copy the image, the ASPX page and its corresponding code behind file:

- GoogleNews.gif
- GoogleNews.aspx
- GoogleNews.aspx.cs

Google News (con't)

Screenshots



News articles regarding the current account (in this example, the name of the account is 'Microsoft')

Grid Preferences

Overview

The goal of this sample is to demonstrate how the GridPreferences control can be used in conjunction with a Grid to allow the end-user to select the fields he or she wants to see. Furthermore, this sample also demonstrates how to preserve user preference and how to subsequently retrieve them.

This sample demonstrates the following:

- Use a Save dialog page to present the GridPreferences control
- Read metadata about an entity and display the list of fields in the GridPreferences control.
- Save the user selections
- Use an AreaPage to display a standard grid
- Retrieve the user preferences and display only the selected fields in the Grid

Installation

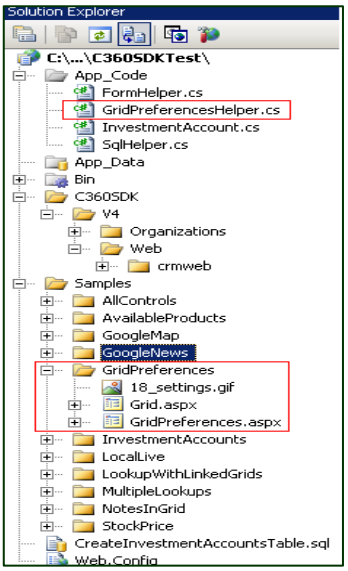
Step 1 – Edit ISV.config

This sample must be added as a new link under a "top level" menu (i.e.: a pull down menu on the top menu bar at the top of the main CRM window). Therefore we need to edit the "CustomMenus" section in Microsoft CRM's ISV.config file like so:

```
  <Menu Title="Testing c360 SDK">
    <MenuItem Title="Grid Preferences"
Url="http://localhost:5517/c360SdkTest/Samples/GridPreferences/Grid.aspx?orgname=<OrganizationName>" WinMode="0"
Client="Web,OutlookWorkstationClient" AvailableOffline="false" />
  </Menu>
```

Note: you will need to adjust the URL in the above sample to match your environment.

Step 2 – Copy files



In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "GridPreferences".

In this new folder, copy the image, the two ASPX page and their corresponding code behind files:

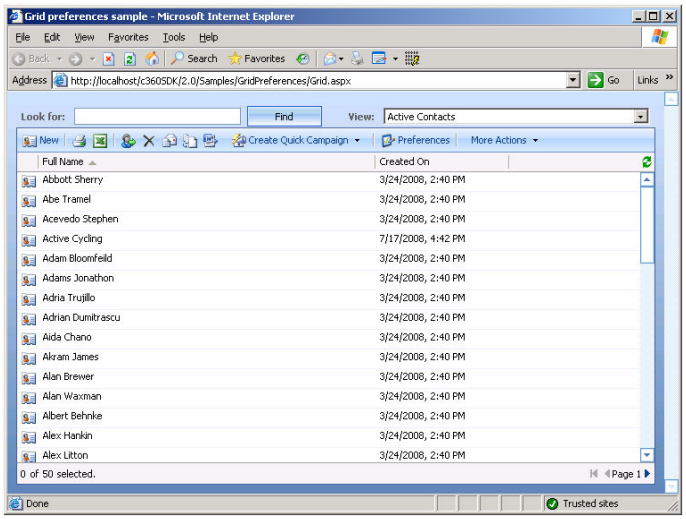
- 18_settings.gif
- Grid.aspx
- Grid.aspx.cs
- GridPreferences.aspx
- GridPreferences.aspx.cs

There is also one file included in this sample that must be copied to the App_Code folder in your solution:

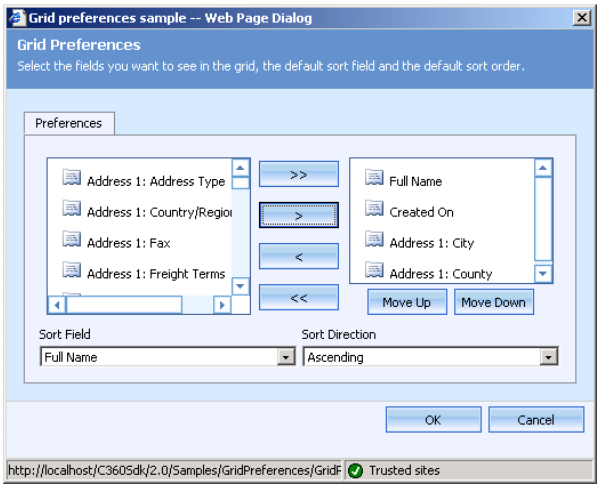
- GridPreferences.cs

Grid Preferences (con't)

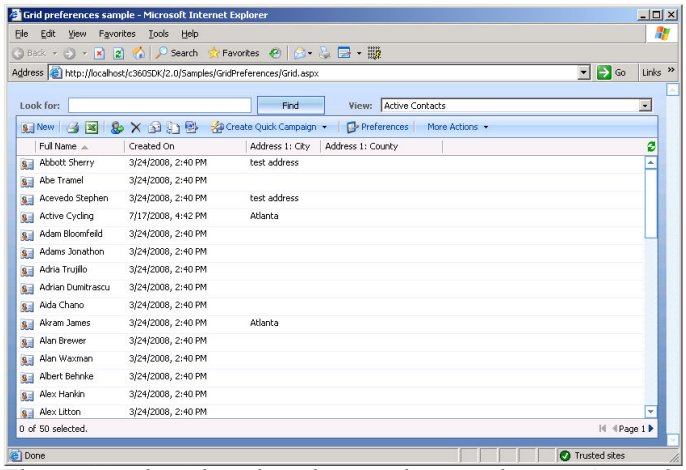
Screenshots



A standard grid with 2 default fields



The preferences screens where the user can select fields



The original grid updated according to the user's preferences

Investment Accounts

Overview

The goal of this sample is manage “investment accounts” related to CRM contacts. This information is maintained in an external database and we want to display the data in a grid at the contact level. We want to create new record, delete and update existing ones.

This sample demonstrates the following:

- List records from an external database in a grid
- Add a “CustomAction” to a grid (the “delete” action)
- Use an “ActionPage” to implement the custom action
- Implement a custom “DoubleClickHandler” in a grid
- Use an “Edit” page to create new records and update existing ones

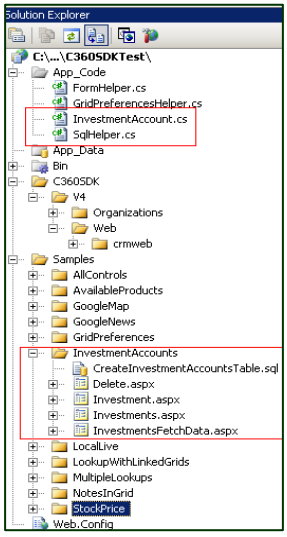
Installation

Step 1 – Edit ISV.config

This sample must be added as a new link on the left navigation bar of the Microsoft CRM contact edit screen. Therefore we need to edit the “contact” entity section in Microsoft CRM’s ISV.config file like so:

```
    <Entity name="contact">
            <NavBar ValidForCreate="0" ValidForUpdate="1">
                    <NavBarItem Icon="/_imgs/ico_16_1055_d.gif"
    Url="http://localhost:1465/c360SdkTest/Samples/InvestmentAccounts/InvestmentAccounts.aspx?orgname=<OrganizationName>
"                                   Id="investmentAccounts" >
                                    <Titles>
                                            <Title LCID="1033" Text="Investment Accounts" />
                                    </Titles>
                    <NavBarItem>
            </NavBar>
    </Entity>
```

Step 2 – Copy files



In your Visual Studio 2005 project, make sure you have a folder called “Samples” and create a sub-folder called “InvestmentAccounts”.

In this new folder, copy the four ASPX pages and their corresponding code behind files:

- Delete.aspx
- Investment.aspx
- Investments.aspx
- InvestmentsFetchData.aspx

There are also three files included in this sample that must be copied to the App_Code folder in your solution:

- InvestmentAccount.cs
- SqlHelper.cs

Investment Accounts (con't)

Step 3 – Create the sample database

For your convenience, a SQL script called "CreateSampleDatabase.sql" is included with our samples to help you create the database but you can also choose to create it manually if you prefer. If you decide to create it manually, make sure to name it "c360SdkSamples".

Step 4 – Create the table in the sample database

For you convenience, we have included a SQL script file called "CreateInvestmentAccountsTable.sql" to create this table and all the required fields.

Step 5 – Update the connection string

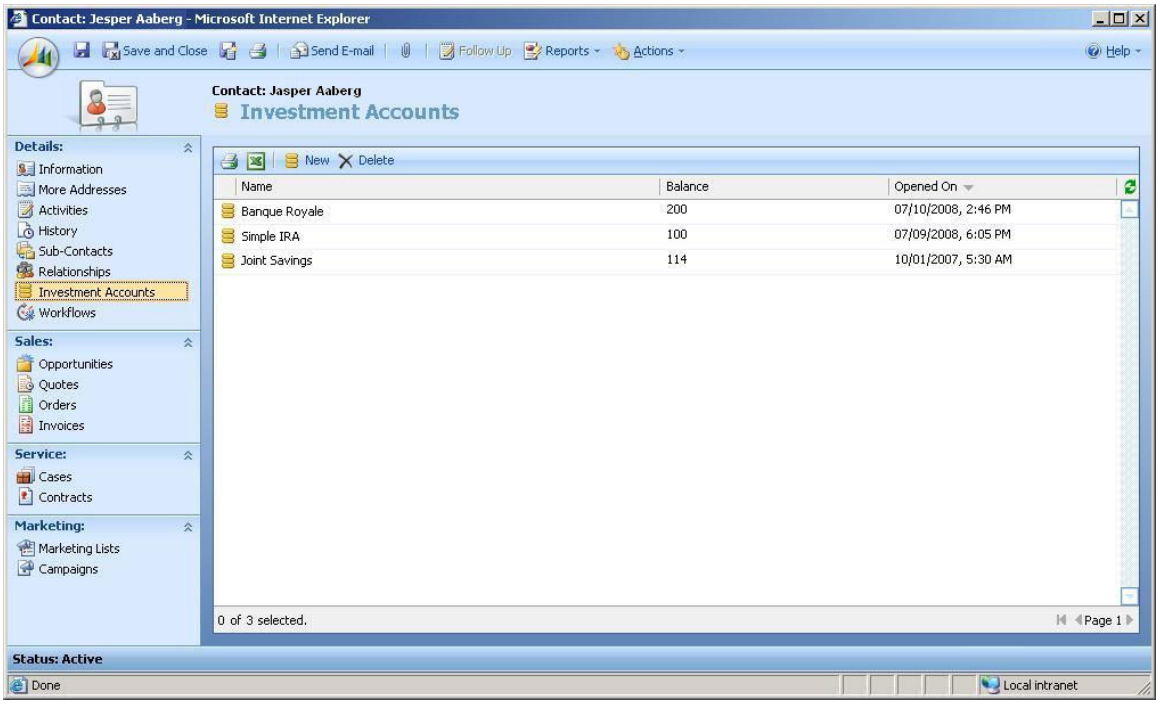
In your project's web.config add the following:

```
<connectionStrings>
        <add name="c360SdkSamples"
                connectionString="Data Source=(local);database=c360SdkSamples;Integrated Security=True;"
                providerName="System.Data.SqlClient" />
</connectionStrings>
```

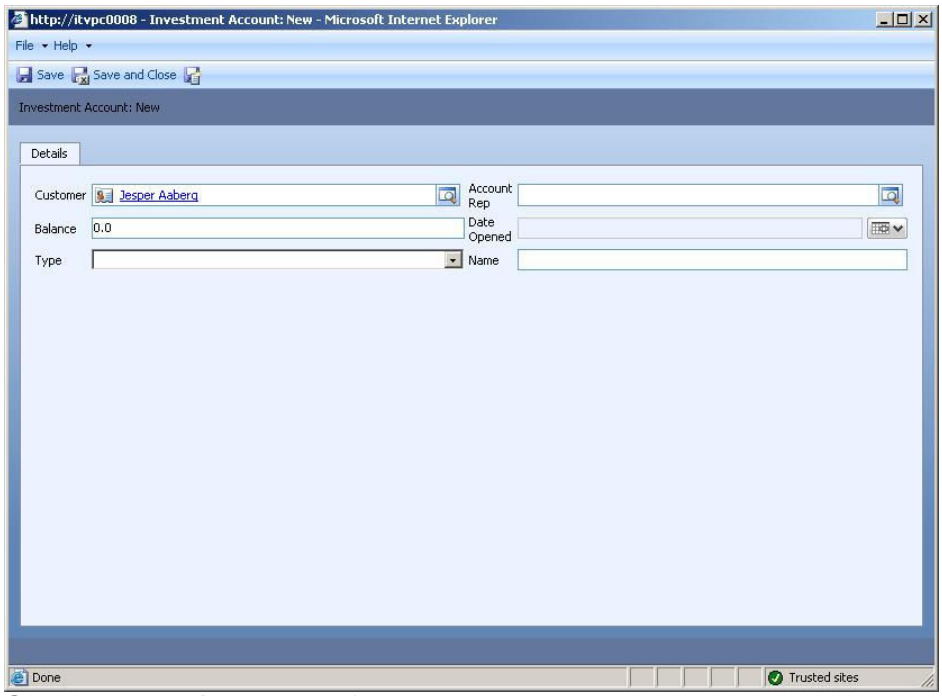
Make sure to edit this connection string to match your environment.

Investment Accounts (con't)

Screenshots



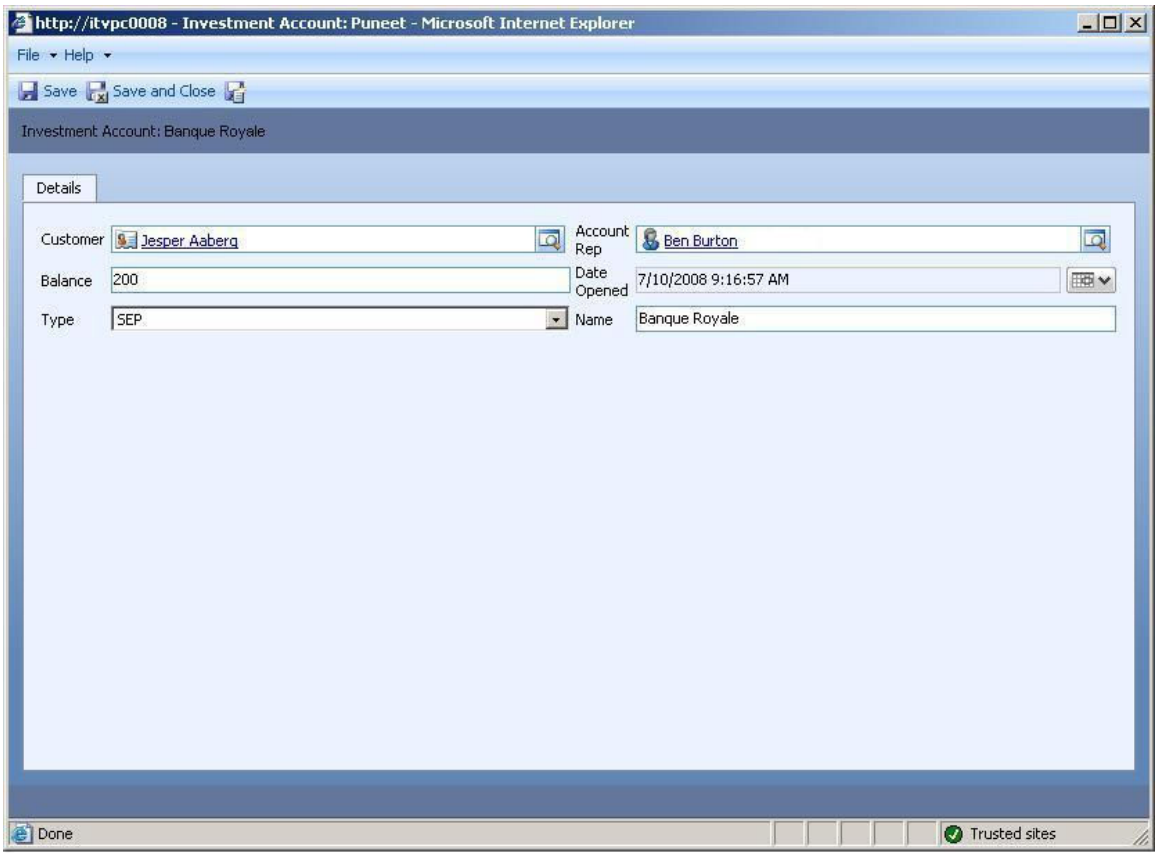
List of Investment accounts linked to a CRM contact (the delete button is a custom action)



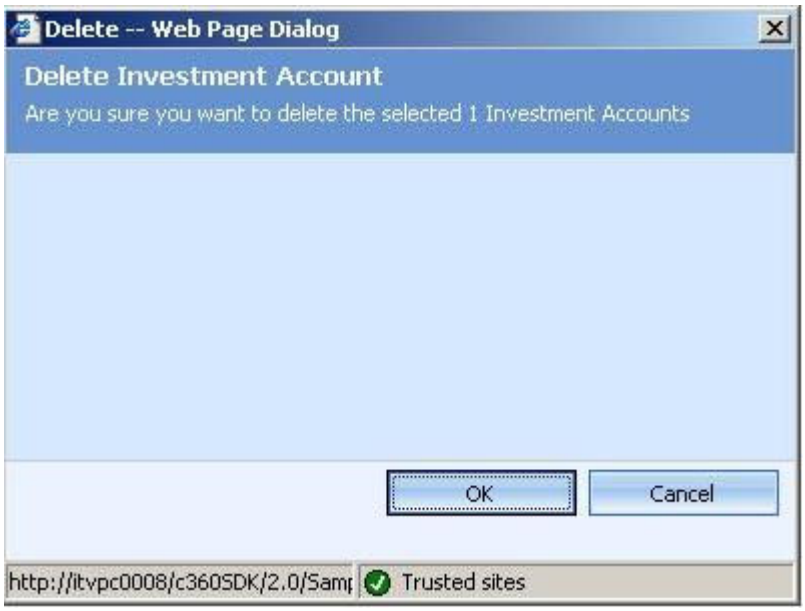
Creating a new Investment Account

Investment Accounts (con't)

Screenshots (con't)



Editing an existing investment account



Deleting an investment account

Local Live Map

Overview

This sample mimics the Google map sample but it uses Microsoft's Local Live maps instead. It displays a map of the area where an account or contact is located and, in the case of an order, it displays directions to get to the delivery destination. This sample was developed by Pim Klingens.

This sample demonstrates the following:

- Fetch data about the current account or order
- Redirect the user to Local Live map and pass the information expected by local Live in order to be able to display a map

Installation

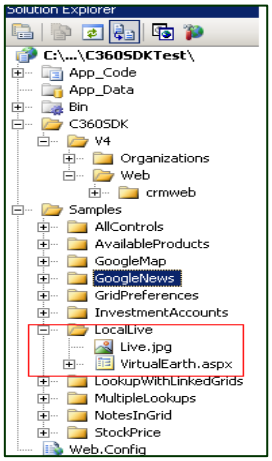
Step 1 – Edit ISV.config

This sample must be added as a new link on the left navigation bar of the Microsoft CRM account edit screen and the order edit screen as well. Therefore we need to edit the "account" and the "salesorder" entity sections in Microsoft CRM's ISV.config file like so:

```
  <Entity name="account">
    <NavBar ValidForCreate="0" ValidForUpdate="1">
      <NavBarItem Icon="http://localhost:4589/c360SdkTest/Samples/LocalLive/Live.jpg" Title="Local Live map"
Url="http://localhost:4589/c360SdkTest/Samples/LocalLive/VirtualEarth.aspx?orgname=<OrganizationName>" Id="localLiveMap"
/>
    </NavBar>
  </Entity>
  <Entity name="salesorder">
    <NavBar ValidForCreate="0" ValidForUpdate="1">
      <NavBarItem Icon="http://localhost:4589/c360SdkTest/Samples/LocalLive/Live.jpg" Title="Local Live map"
Url="http://localhost:4589/c360SdkTest/Samples/LocalLive/VirtualEarth.aspx?orgname=<OrganizationName>" Id="localLiveMap"
/>
    </NavBar>
  </Entity>
```

Note: you will need to adjust the URL in the above sample to match your environment.

Step 2 – Copy files



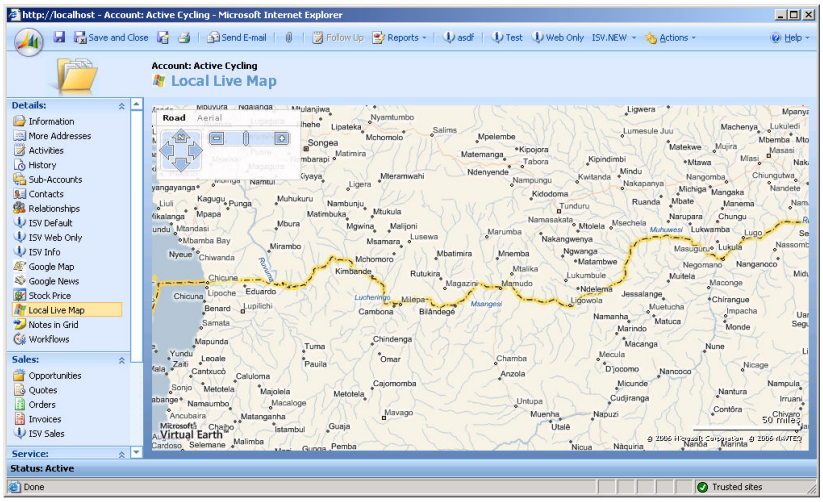
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "LocalLive".

In this new folder, copy the image, the ASPX page and its corresponding code behind file:

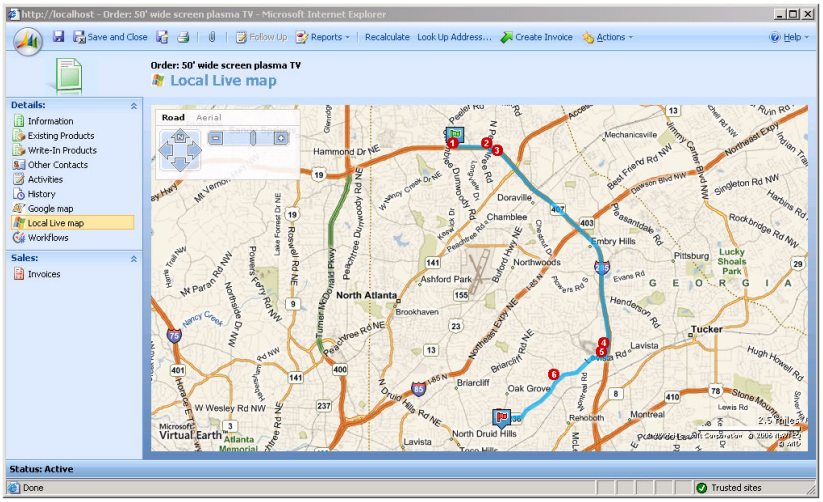
- Live.jpg
- VirtualEarth.aspx
- VirtualEarth.aspx.cs

Local Live Map (con't)

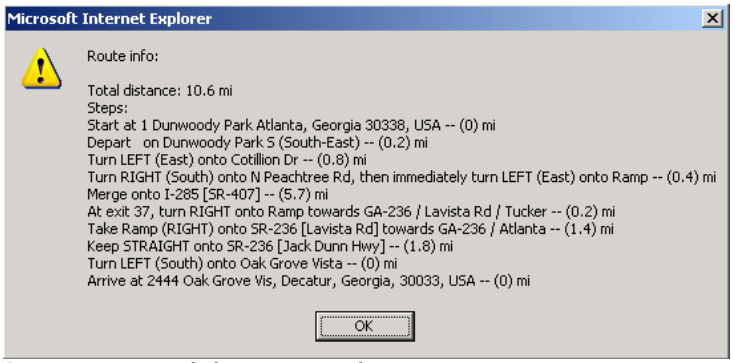
Screenshots



Map showing where an account is located



Map showing directions where an order must be delivered



Directions to deliver an order

Stock Price

Overview

The goal of this sample is to display stock market information for publicly traded companies.

This sample demonstrates the following:

- Fetch the stock symbol of the current account
- Redirect the user to Google finance and pass the information expected by Google in order to be able to display the financial information.

Installation

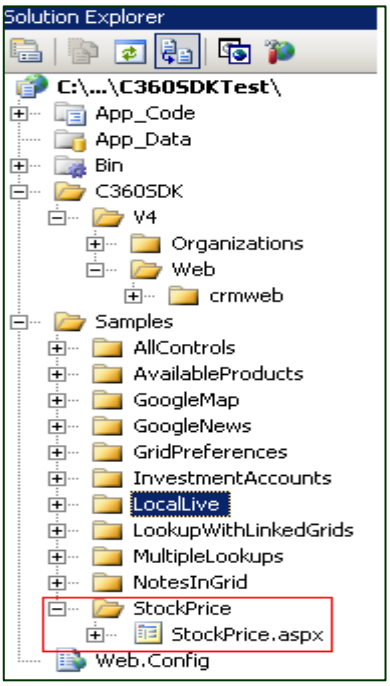
Step 1 – Edit ISV.config

This sample must be added as a new link on the left navigation bar of the Microsoft CRM account edit screen. Therefore we need to edit the "account" entity section in Microsoft CRM's ISV.config file like so:

```
<Entity name="account">
  <NavBar ValidForCreate="0" ValidForUpdate="1">
    <NavBarItem Icon=" /_imgs/ico_16_salesprocess.gif" Title="Stock Price"
Url="http://localhost:4589/c360SdkTest/Samples/StockPrice/StockPrice.aspx?orgname=<OrganizationName>" Id="stockPrice" />
  </NavBar>
</Entity>
```

Note: you will need to adjust the URL in the above sample to match your environment.

Step 2 – Copy files



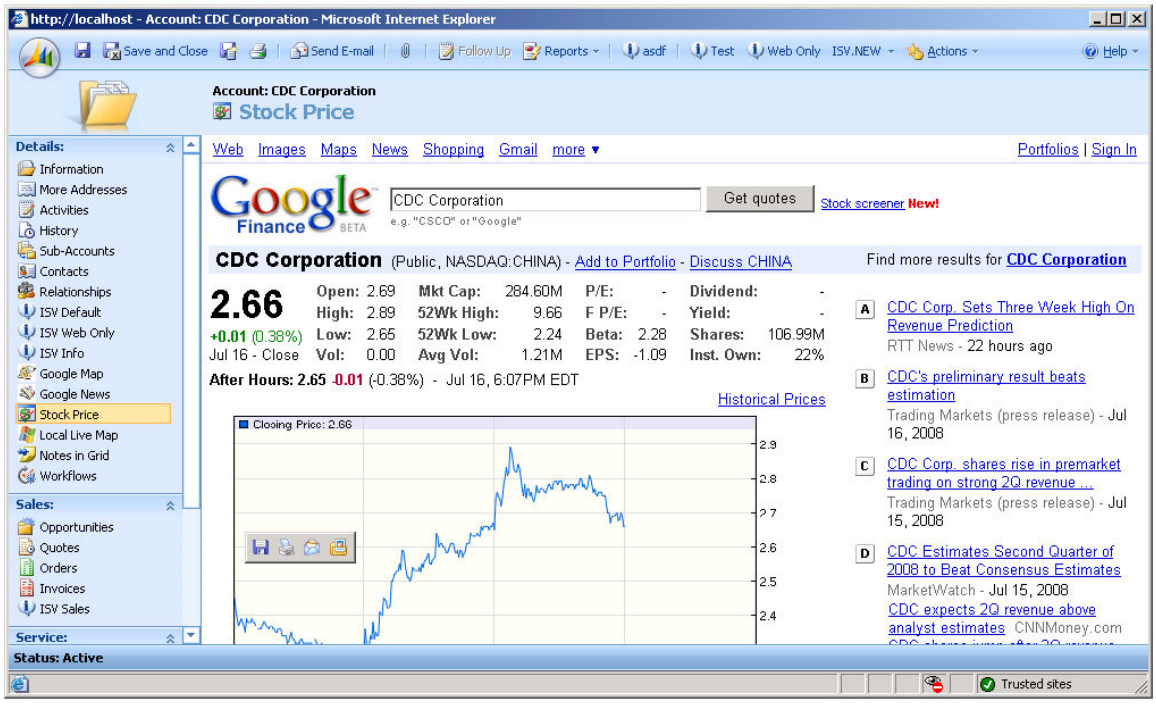
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "StockPrice".

Copy the ASPX page and its corresponding code behind file in this new folder:

- StockPrice.aspx
- StockPrice.aspx.cs

Stock Price (con't)

Screenshots



Stock price information

Lookup with Linked Grids

Overview

The goal of this sample is to display how the look up control can be used in conjunction with grid control to allow the user to select the fields he or she wants to see.

This sample demonstrates the following:

- Lookup control to select the Territory.
- Fetches all the accounts in the selected territory and displays in the first Grid.
- Fetches all the contacts releated to the selected account in the first grid and displays it in the second grid.

Installation

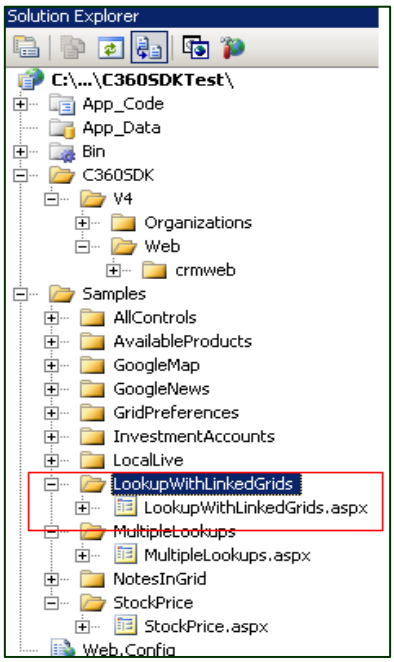
Step 1 – Edit ISV.config

This sample must be added as a new link under a "top level" menu (i.e.: a pull down menu on the top menu bar at the top of the main CRM window). Therefore we need to edit the "CustomMenus" section in Microsoft CRM's ISV.config file like so:

```
<Menu Title="Testing c360 SDK">
  <MenuItem Title="Lookup with linked Grids" Url="http://localhost:5517/c360SdkTest/Samples/LookupWithLinkedGrids/
LookupWithLinkedGrids.aspx?orgname=<OrganizationName>" WinMode="0" Client="Web,OutlookWorkstationClient"
AvailableOffline="false" />
</Menu>
```

Note: you will need to adjust the URL in the above sample to match your environment.

Step2: Copy the required files.



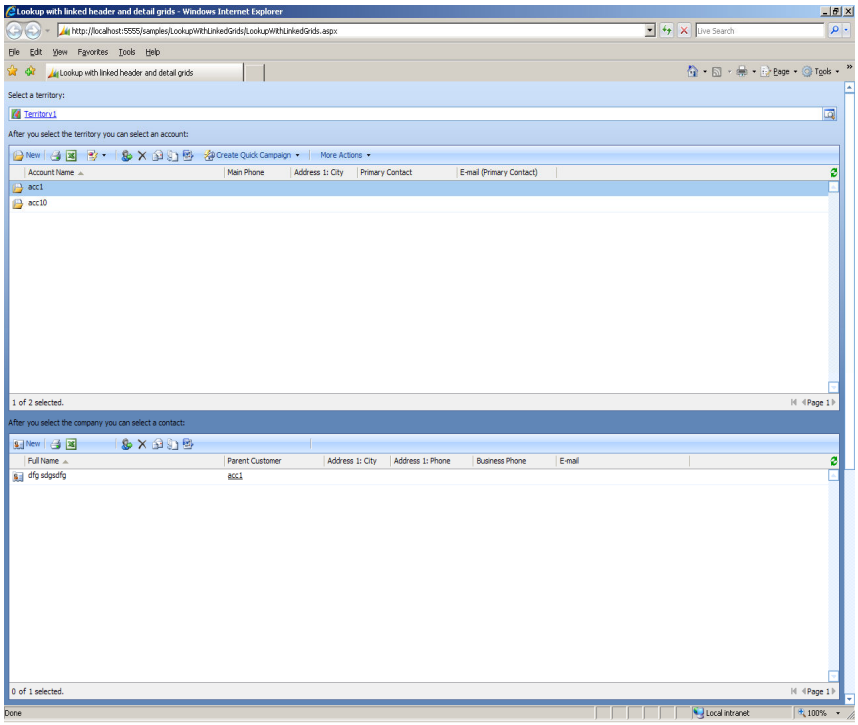
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "LookupWithLinkedGrids".

Copy the ASPX page and its corresponding code behind file in this new folder:

- LookupWithLinkedGrids.aspx
- LookupWithLinkedGrids.aspx.cs

Look up with Linked Grids (Con't)

Screenshots



Look up with linked grids

Mulitple Lookups

Overview

The goal of this sample is to display how multiple lookup controls can be used and linked to each other to allow the user to select the fields he or she wants to see.

This sample demonstrates the following:

- Lookup control to select the Territory.
- Only after the data for first lookup is selcted the second lookup is enabled to select the account records.
- Once the data is selected in the second lookup the third lookup is enabled and used to select any contact record.

Installation

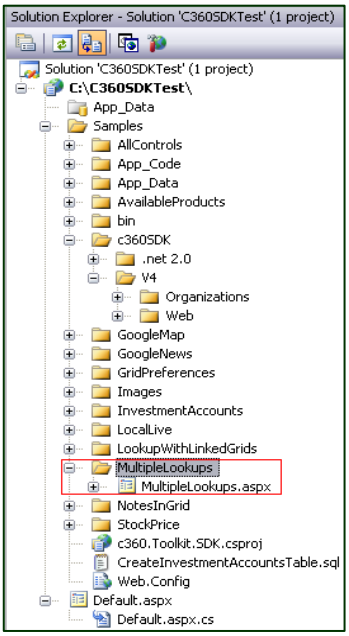
Step 1 – Edit ISV.config

This sample must be added as a new link under a "top level" menu (i.e.: a pull down menu on the top menu bar at the top of the main CRM window). Therefore we need to edit the "CustomMenus" section in Microsoft CRM's ISV.config file like so:

```
<Menu Title="Testing c360 SDK">
  <MenuItem Title="Lookup with linked Grids" Url="http://localhost:5517/c360SdkTest/Samples/MultipleLookups/
MultipleLookups.aspx?orgname=<OrganizationName>" WinMode="0" Client="Web,OutlookWorkstationClient"
AvailableOffline="false" />
</Menu>
```

Note: you will need to adjust the URL in the above sample to match your environment.

Step2: Copy the required files.



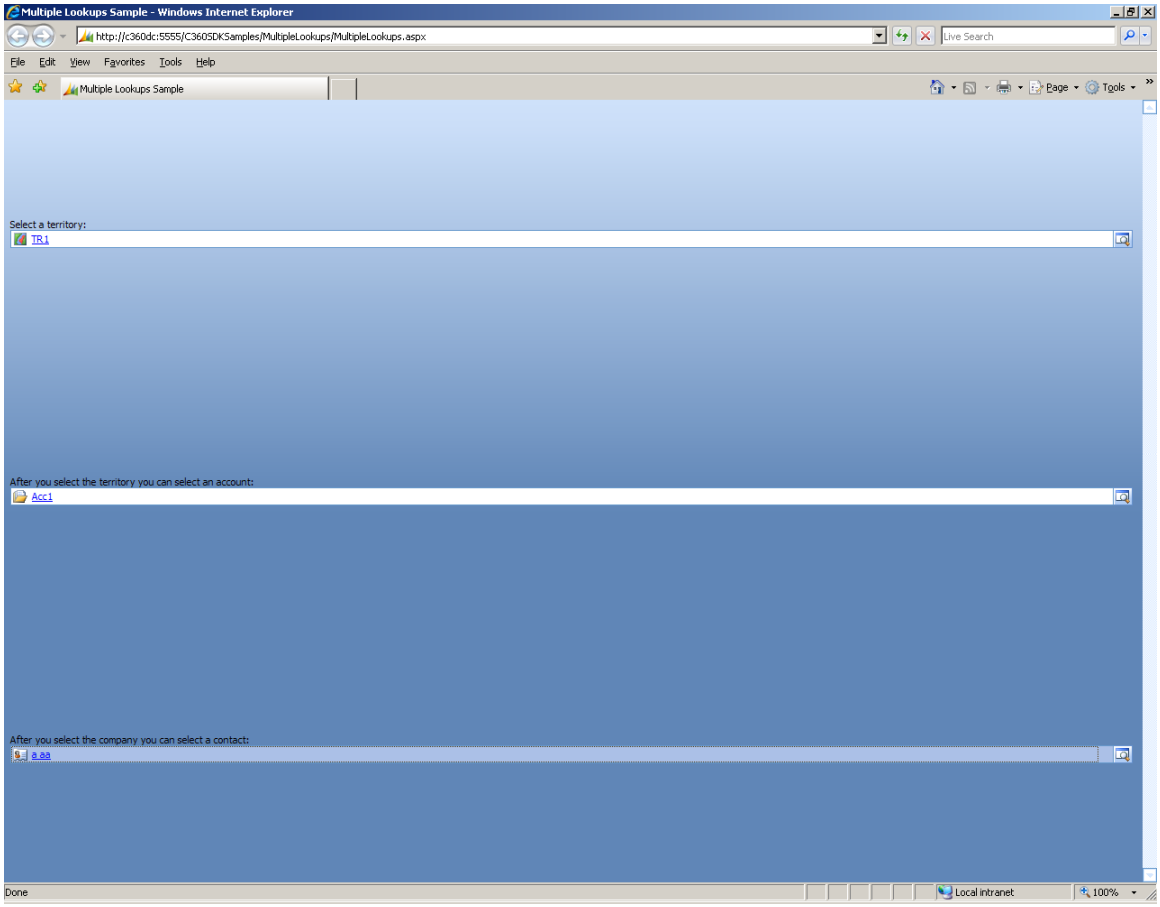
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "MultipleLookups".

Copy the ASPX page and its corresponding code behind file in this new folder:

- MultipleLookups.aspx
- MultipleLookups.aspx.cs

Multiple Lookups (Con't)

Screenshots



Multiple look ups screen

Notes in Grid

Overview

The goal of this sample is to display how the grid control can be used in the crm entity forms.

This sample demonstrates the following:

- Fetches all the notes available for the entity and displays in the grid.

Installation

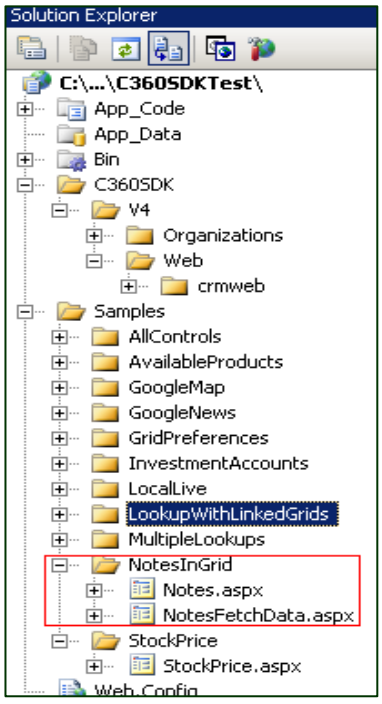
Step 1 – Edit ISV.config

This sample must be added as a new link on the left navigation bar of the Microsoft CRM account edit screen. Therefore we need to edit the "account" entity section in Microsoft CRM's ISV.config file like so:

```
  <Entity name="account">
    <NavBar ValidForCreate="0" ValidForUpdate="1">
      <NavBarItem Icon=" /_imgs/ico_16_5.gif" Title="Notes in Grid"
Url="http://localhost:4589/c360SdkTest/Samples/NotesInGrid/Notes.aspx?orgname=<OrganizationName>" Id=" NotesInGrid " />
    </NavBar>
  </Entity>
```

Note: you will need to adjust the url in the above sample to match your environment.

Step 2 – Copy files



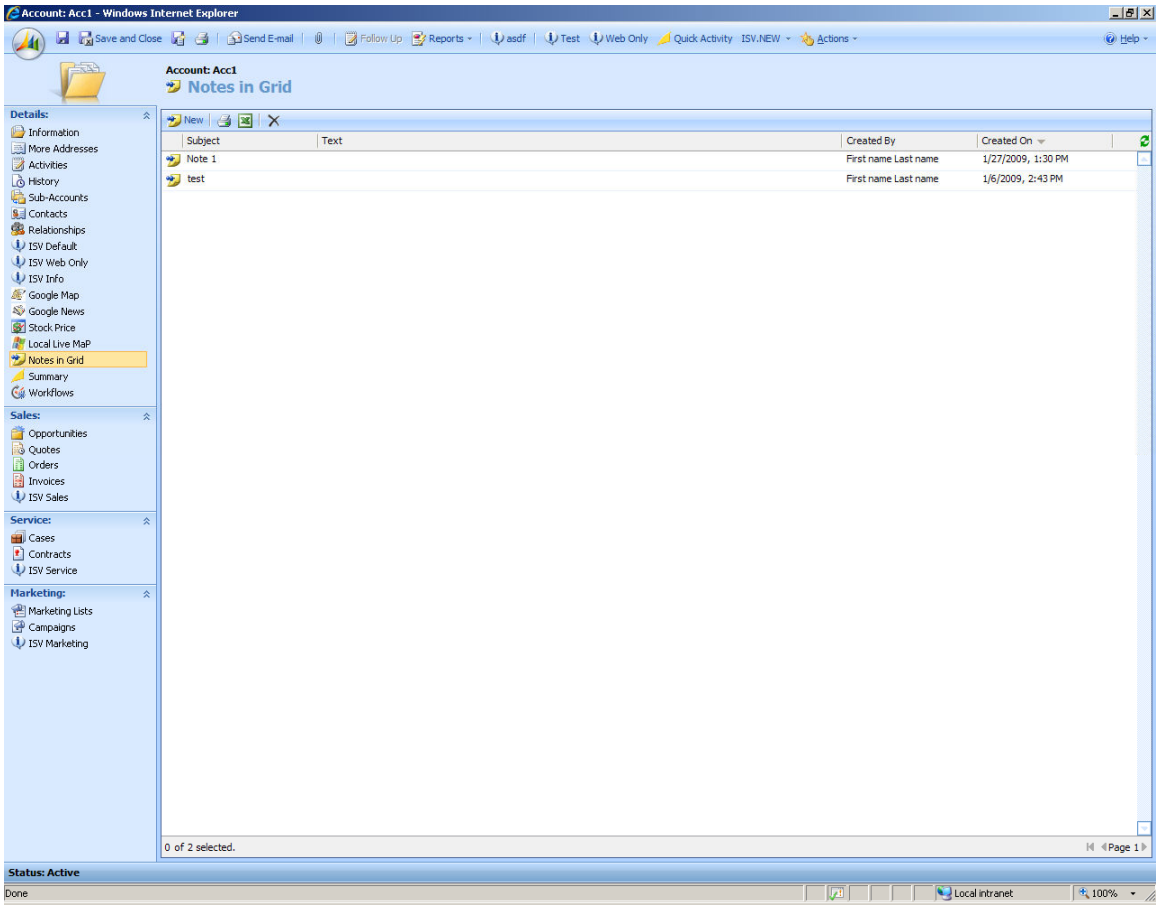
In your Visual Studio 2005 project, make sure you have a folder called "Samples" and create a sub-folder called "NotesInGrid".

Copy the ASPX page and its corresponding code behind file in this new folder:

- Notes.aspx
- NotesFetchData.aspx
- Notes.aspx.cs
- NotesFetchData.aspx.cs

Notes in Grid (Con't)

Screen shots



Notes displayed in grid

Editable Grid

Overview

The goal of this sample is display the grid control in the editable mode.

This sample demonstrates the following:

- Fetches all the active accounts and displays in the grid.

Installation

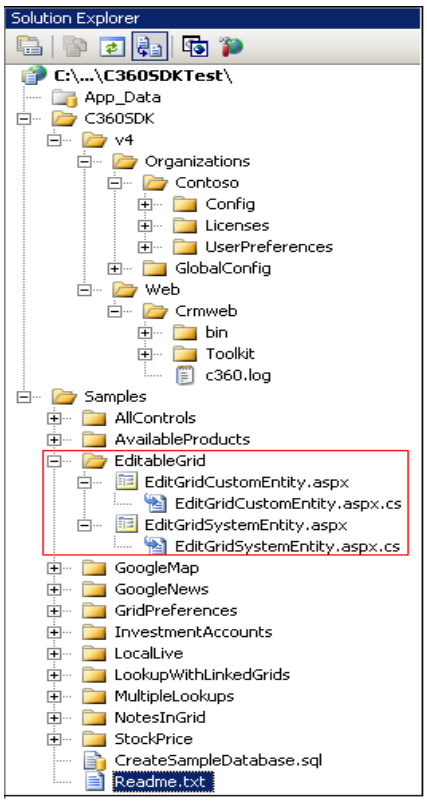
Step 1 – Edit ISV.config

This sample must be added as a new link under a “top level” menu (i.e.: a pull down menu on the top menu bar at the top of the main CRM window). Therefore we need to edit the “CustomMenus” section in Microsoft CRM’s ISV.config file like so:

```
  <Menu Title="Testing c360 SDK">
    <MenuItem Title="All Controls" Url="http://localhost:5517/c360SdkTest/Samples/
EditableGrid/EditGridSystemEntity.aspx?orgname=<OrganizationName>" WinMode="0" Client="Web,OutlookWorkstationClient"
AvailableOffline="false" />
  </Menu>
```

Note: you will need to adjust the URL in the above sample to match your environment

Step 2 – Copy files



In your Visual Studio 2005 project, make sure you have a folder called “Samples” and create a sub-folder called “EditableGrid”.

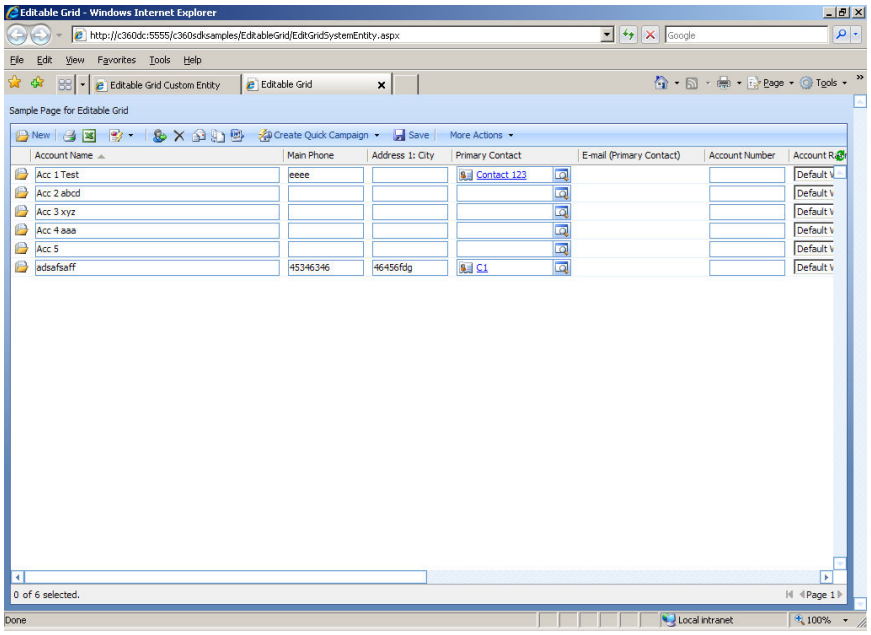
In this new folder, copy the ASPX page and its corresponding code behind file:

- EditGridSystemEntity.aspx
- EditGridSystemEntity.aspx.cs
- EditGridCustomEntity.aspx
- EditGridCustomEntity.aspx.cs

Editable Grid (Con't)

Screen shots

Editable Grid for Account entity



Editable Grid for Custom entity